# Infrastructure for Parallel Adaptive Unstructured Mesh Simulations


*M.S. Shephard, C.W. Smith, E.S. Seol, D.A. Ibanez, Q. Lu, O. Sahni, M.O. Bloomfield, B.N. Granzow*

Scientific Computation Research Center, Rensselaer Polytechnic Institute

*G. Hansen, K. Devine and V. Leung*

Sandia National Laboratories

*K.E. Jansen, M. Rasquin and K.C. Chitale*

University of Colorado

*M.W. Beall and S. Tendulkar*

Simmetrix Inc.

# Presentation Outline

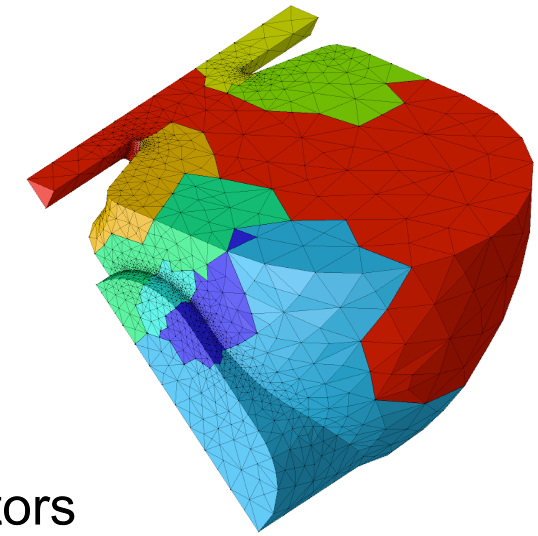
Meshes of multi-million element meshes needed even with the use of adaptive methods

- Simulations must be run on massively parallel computers with information (mesh) distributed at all times
- Need an effective parallel mesh infrastructure and associated utilities to deal with the mesh and its adaptation

Presentation outline

- Unstructured meshes on massively parallel computers
  - Representations and support of a distributed mesh
  - Dynamic load balancing
  - Mesh adaptation using parallel mesh modification
- Component-based infrastructure for parallel adaptive analysis
- Albany computational mechanics environment and testbed
- Comments on hand-on session materials

# Parallel Adaptive Analysis



## Components

- Scalable FE or FV analysis
  - Form the system of equations
  - Solve the system of equations
- Parallel unstructured mesh infrastructure
  - Including means to move entities
- Mesh adaptation procedure
  - Driven by error estimates and/or correction indicators
  - Maintain geometric fidelity
  - Support analysis needs (e.g., maintain boundary layer structure)
- Dynamic load balancing
  - Rebalance as needed
  - Support predictive methods to control memory use and/or load
  - Fast partition improvement (considering multiple entities)

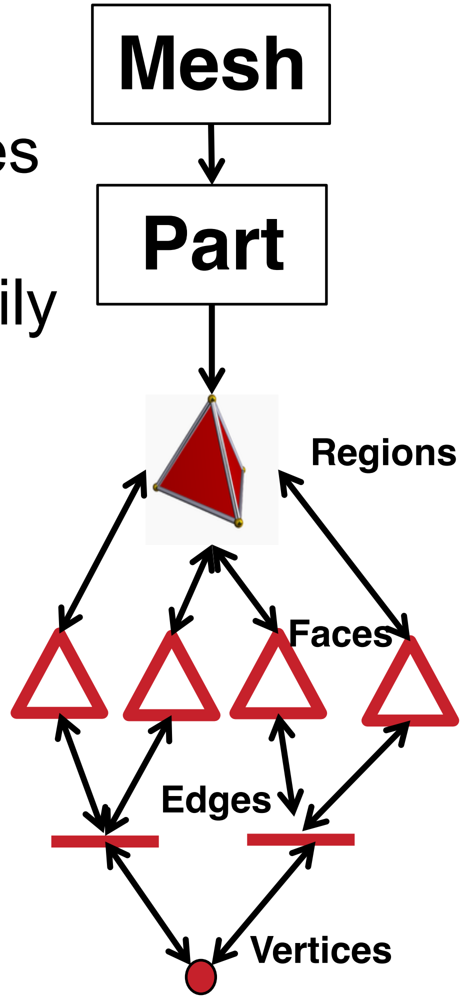
## All components must operate in parallel

- Scalability requires using same parallel control structure for all steps – partitioned mesh

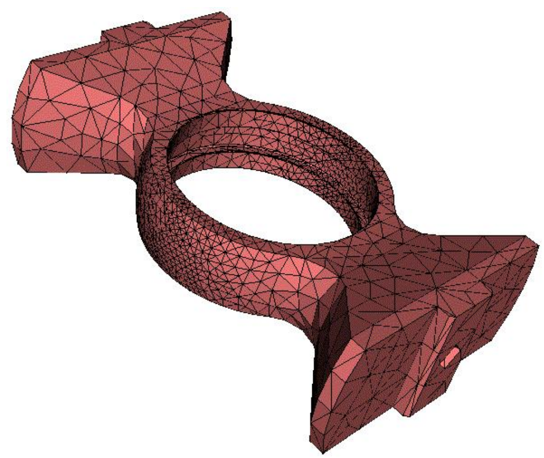
# Background

## Geometry-Based Analysis

- Geometry, Attribute: analysis domain
- Mesh: 0-3D topological entities and adjacencies
- Field: distribution of solution over mesh
- Common requirements: data traversal, arbitrarily attachable user data, data grouping, etc.
- Complete representation: store sufficient entities and adjacencies to get any adjacency in O(1) time



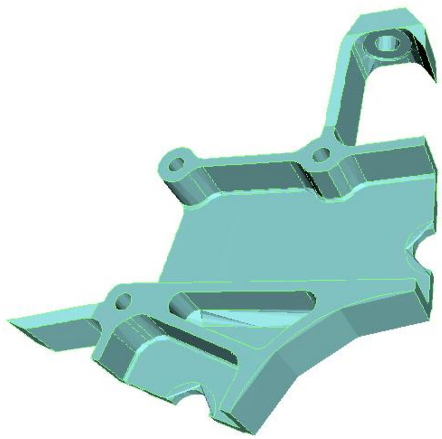

*Geometric model*



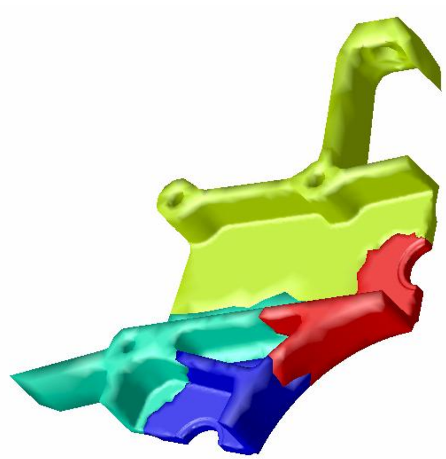
*Mesh*

# Parallel Unstructured Mesh Infrastructure (PUMI)

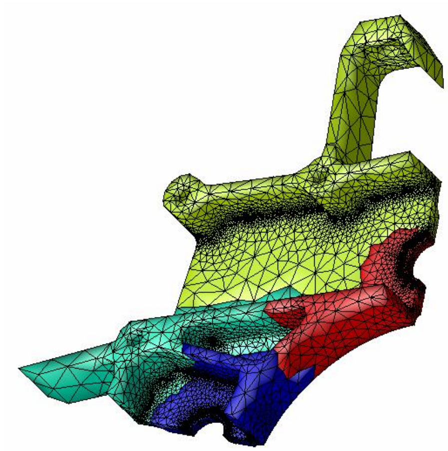
- Capability to partition mesh to multiple parts per process



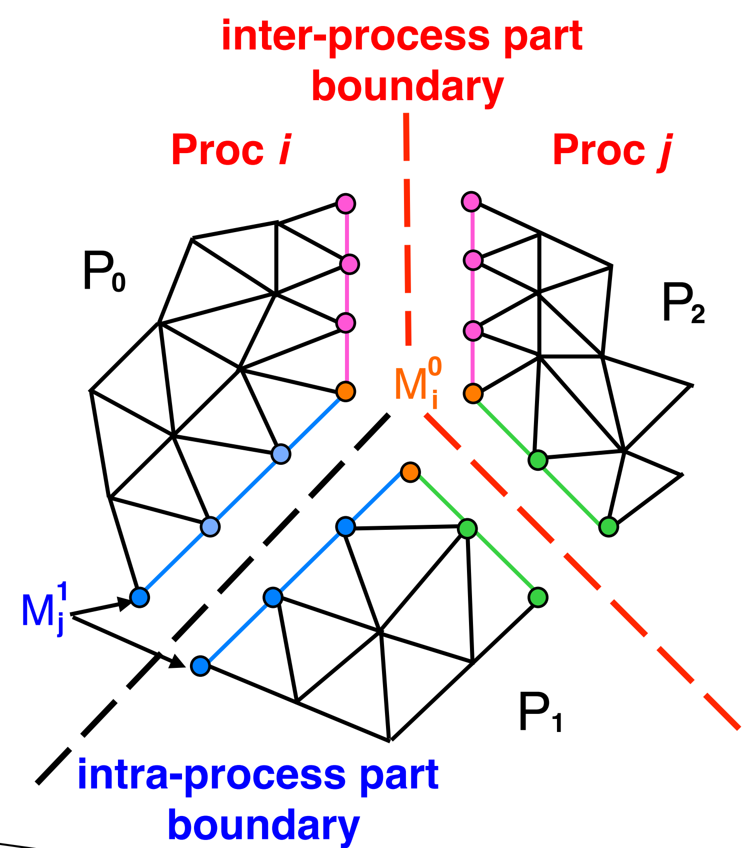
Geometric model



Partition model



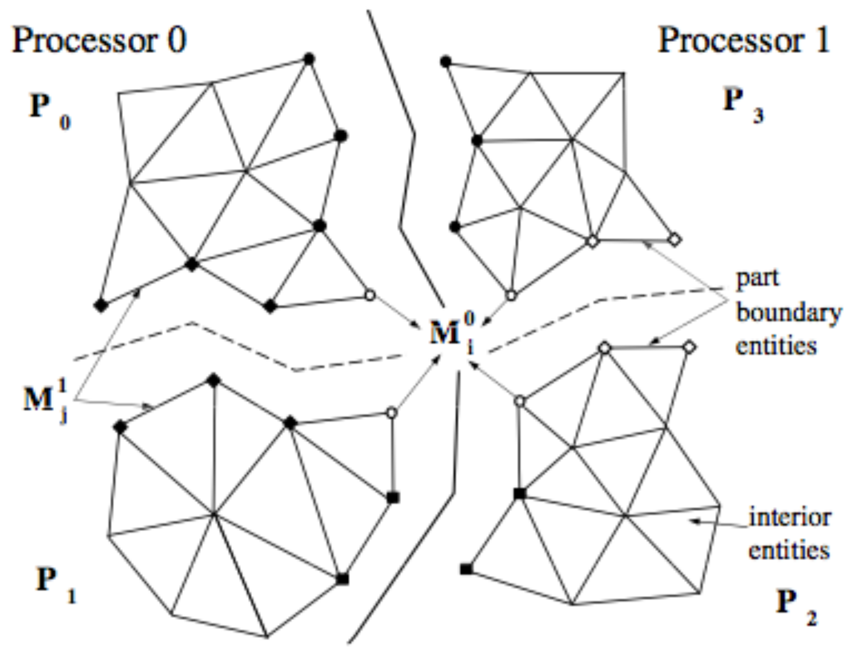
Distributed mesh



# Distributed Mesh Data Structure

Each part $P_i$ assigned to a process

- Consists of mesh entities assigned to $i^{th}$ part.
- Uniquely identified by handle or id plus part number
- Treated as a serial mesh with the addition of *part boundaries*
  - *Part boundary:* groups of mesh entities on shared links between parts
  - *Part boundary entity*: duplicated entities on all parts for which they bound with other higher order mesh entities
  - *Remote copy*: duplicated entity copy on non-local part




FASTMATH

# Mesh Migration

Purpose: Moving mesh entities between parts

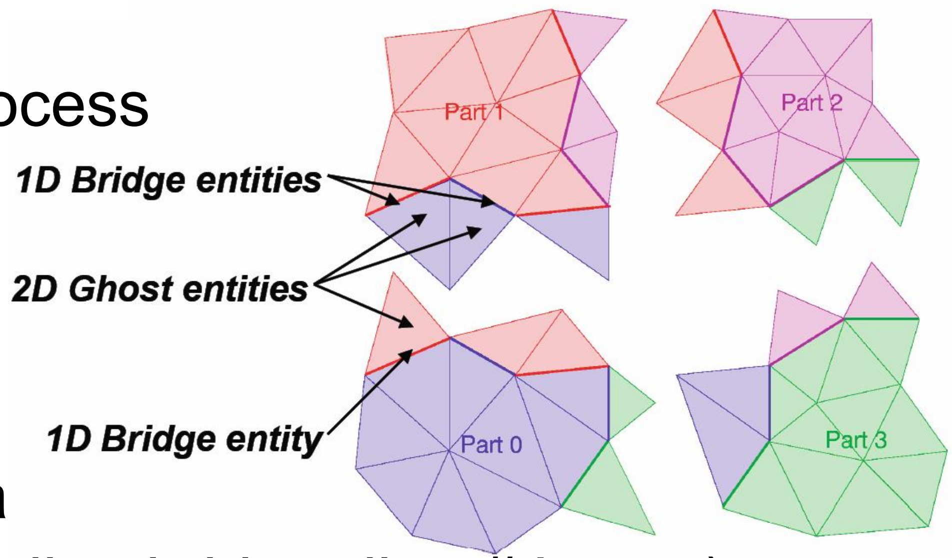
- Dictated by operation - in swap and collapse it's the mesh entities on other parts needed to complete the mesh modification cavity
- Entities to migrate are determined based on adjacencies

Issues

- A function of mesh representation w.r.t. adjacencies, P- set and arbitrary user data attached to them
  - Complete mesh representation can provide any adjacency without mesh traversal - a requirement for satisfactory performance
- Performance issues
  - synchronization, communications, load balance and scalability
  - How to benefit from on-node thread communication (all threads in a processor share the same memory address space)

# Ghosting

- *Goals*: localizing off-part mesh data to avoid inter-process communications for computations
- *Ghost*: read-only, duplicate entity copies not on part boundary including tag data




- *Ghosting rule*: triplet (ghost dim, bridge dim, # layers)
  - Ghost dim: entity dimension to be ghosted
  - Bridge dim: entity dimension used to obtain entities to be ghosted through adjacency
  - # layers: the number of ghost layers measured from the part boundary

E.g, to get two layers of region entities in the ghost layer, measured from faces on part boundary, use ghost_dim=3, bridge_dim=2, and # layers=2

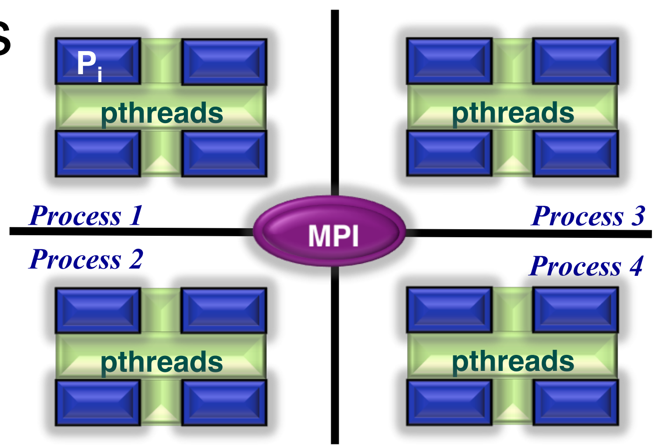
# Two-Level Partitioning to Use MPI and Threads

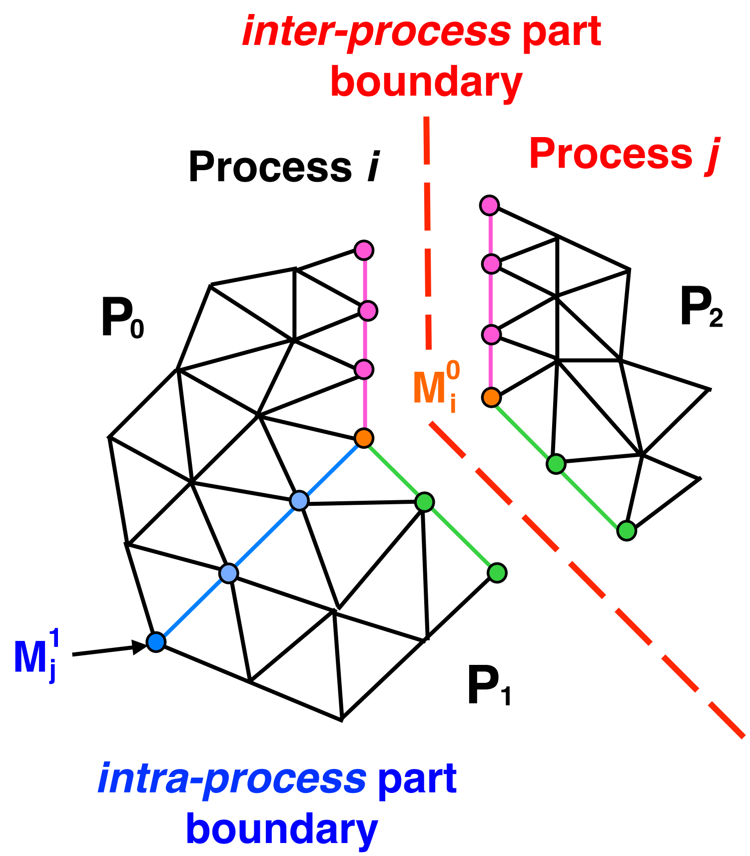
Exploit hybrid architecture of BG/Q, Cray XE6, etc…

- Reduced memory usage

Approach

- Partition mesh to processes, then partition to threads
- Message passing, via MPI, between processes
- Shared memory, via pthreads, within process
- Transparent-to-application use of pthreads



# Blue Gene/Q Two Level Partition Results

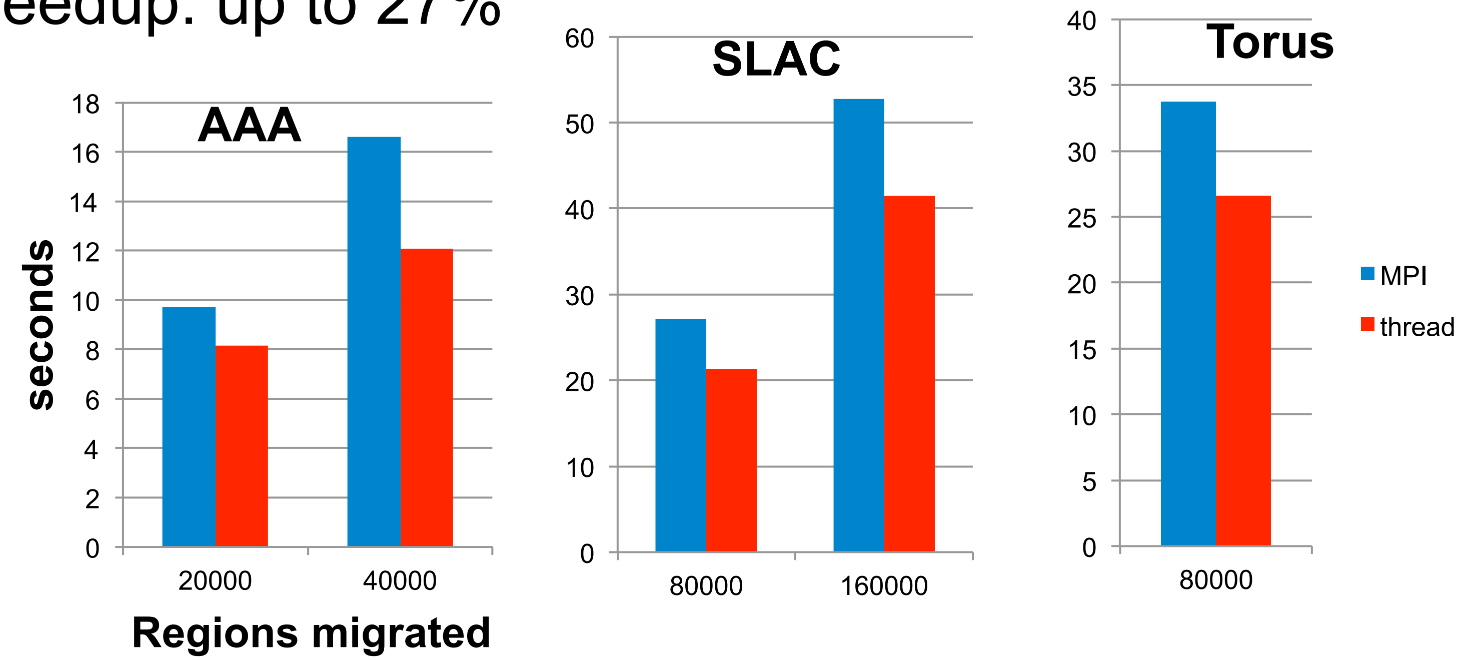
AAA mesh: 2M tets, 32 parts, 2 nodes

SLAC mesh: 17M tets, 64 parts, 4 nodes

Torus mesh: 610M tets, 4096 parts, 256 nodes

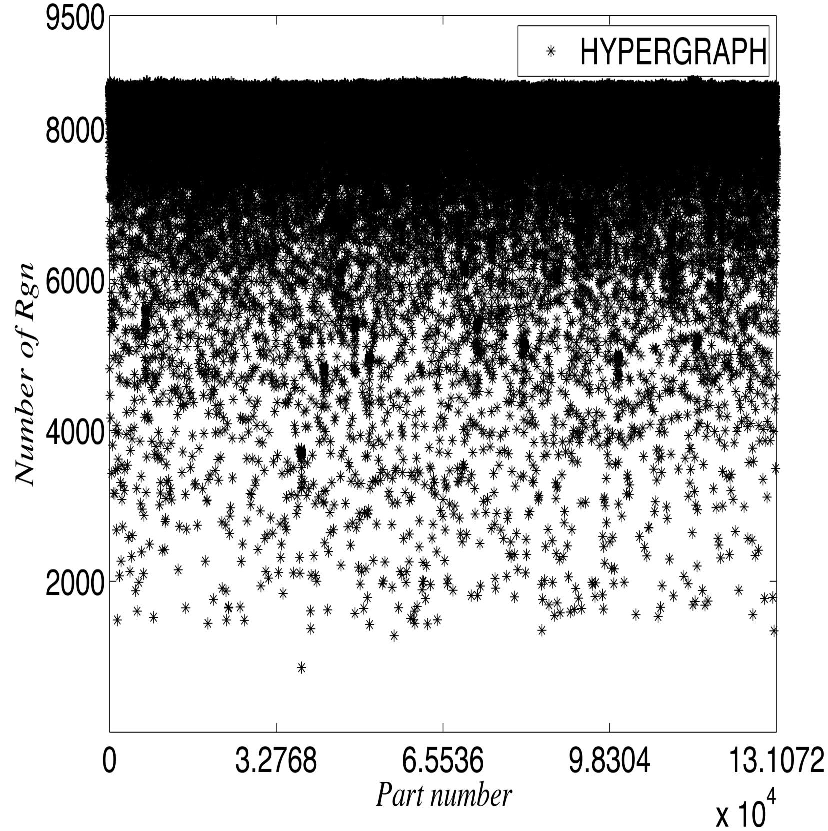
Test: local migration, all MPI vs. 1 MPI rank/16 threads per node

Speedup: up to 27%



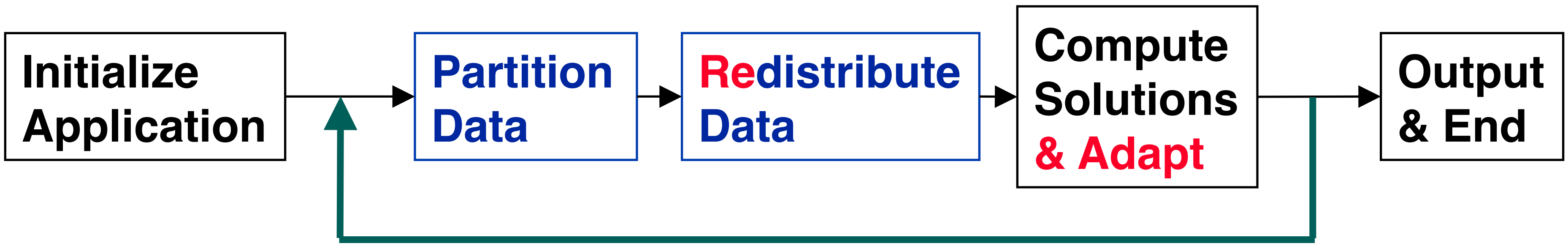
# Dynamic Load Balancing

- Purpose: to rebalance load imbalanced mesh during mesh modification
  - Equal "work load" with minimum inter-process communications
- Two tools being used
  - Zoltan Dynamic Services supporting multiple dynamic partitioners with general control of partition objects and weights
  - ParMa – Partitioning using mesh adjacencies




FASTMATH

# Dynamic Repartitioning (Dynamic Load Balancing)




Dynamic repartitioning (load balancing) in an application:

- Data partition is computed.
- Data are distributed according to partition map.
- Application computes and, perhaps, adapts.
- Process repeats until the application is done.

Ideal partition:

- Processor idle time is minimized.
- Inter-processor communication costs are kept low.
- Cost to redistribute data is also kept low.

# Static vs. Dynamic: Usage and Implementation

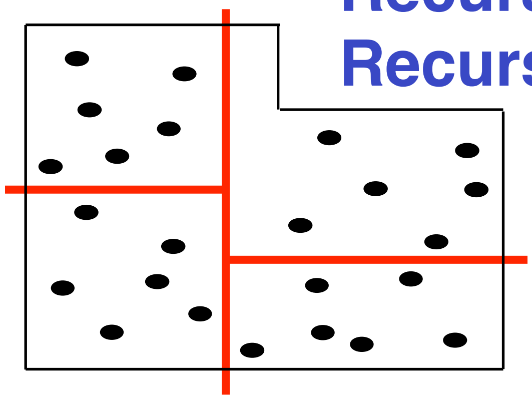
Static:

- Pre-processor to application.
- Can be implemented serially.
- May be slow, expensive.
- File-based interface acceptable.
- No consideration of existing decomposition required.

Dynamic:

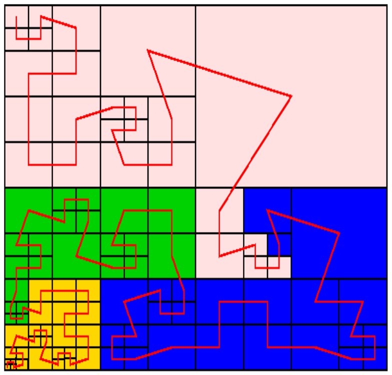
- Must run side-by-side with application.
- Must be implemented in parallel.
- Must be fast, scalable.
- Library application interface required.
- Should be easy to use.
- Incremental algorithms preferred.
  - Small changes in input result small changes in partitions.
  - Explicit or implicit incrementally acceptable.

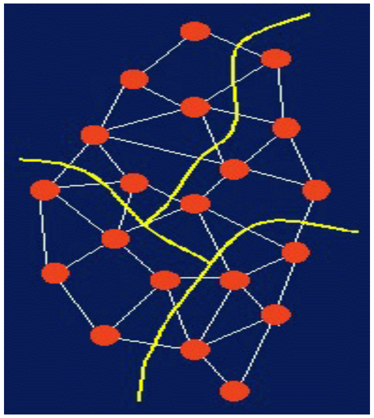
# Zoltan Toolkit: Suite of Partitioners



Recursive Coordinate Bisection (Berger, Bokhari)
Recursive Inertial Bisection (Taylor, Nour-Omid)

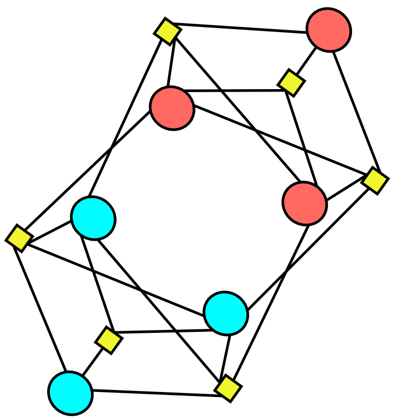
Space Filling Curves
(Peano, Hilbert)
Refinement-tree Partitioning
(Mitchell)





Graph Partitioning
ParMETIS (Karypis, Schloegel, Kumar)
Jostle (Walshaw)

Hypergraph Partitioning & Repartitioning
(Catalyurek, Aykanat, Boman, Devine, Heaphy, Karypis, Bisseling)
PaToH (Catalyurek)



# Geometric Partitioners

Goal: Create parts containing physically close data.

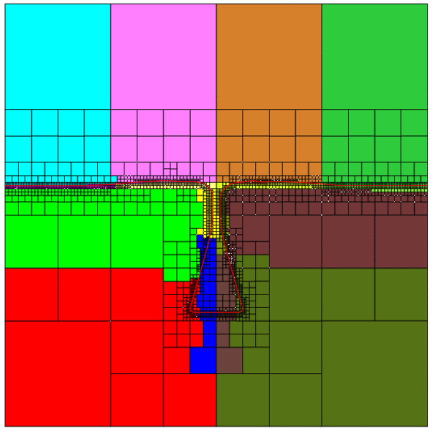
- RCB/RIB: Compute cutting planes that recursively divide work.
- SFC: Partition linear ordering of data given by space-filling curve.

Advantages:

- Conceptually simple; fast and inexpensive.
- Effective when connectivity info is not available (e.g., in particle methods).
- All processors can inexpensively know entire decomposition.
- RCB: Regular subdomains useful in structured or unstructured meshes.
- SFC: Linear ordering may improve cache performance.

Disadvantages:

- No explicit control of communication costs.
- Can generate disconnected subdomains for complex geometries.
- Geometric coordinates needed.



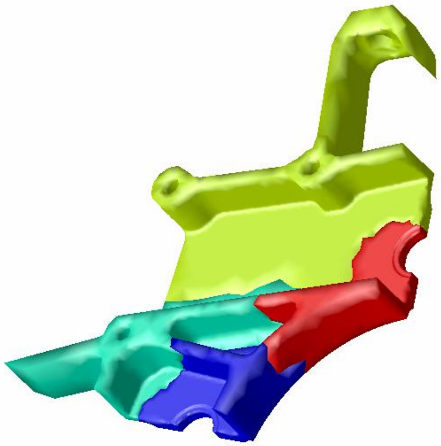
# Topology-based Partitioners

Goal: Balance work while minimizing data dependencies between parts.

- Represent data with vertices of graph/hypergraph
- Represent dependencies with graph/hypergraph edges

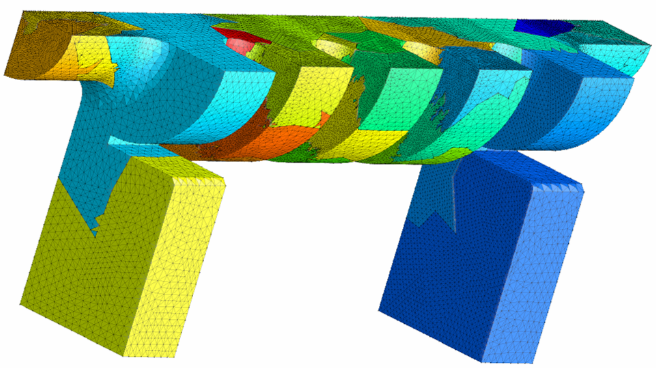
Advantages:

- High quality partitions for many applications
- Explicit control of communication costs
- Much software available
  - Serial: Chaco, METIS, Scotch, PaToH, Mondriaan
  - Parallel: Zoltan, ParMETIS, PT-Scotch, Jostle



Disadvantages:

- More expensive than geometric approaches
- Require explicit dependence info



# Partitioning using Mesh Adjacencies (ParMA)

Mesh and partition model adjacencies represent application data more completely then standard partitioning graph

- All mesh entities can be considered, while graph-partitioning models use only a subset of mesh adjacency information.
- Any adjacency can be obtained in O(1) time (assuming use of a complete mesh adjacency structure).

Advantages

- Directly account for multiple entity types – important for the solve process – most computationally expensive step
- Avoid graph construction
- Easy to use with diffusive procedures

Applications to Date

- Partition improvement to account for multiple entity types – improved scalability of solvers
- Use for improving partitions on really big meshes

# ParMA – Multi-Criteria Partition Improvement

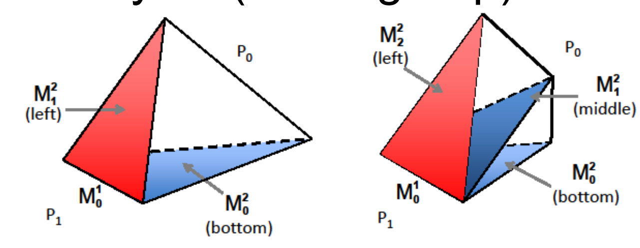
Improved scalability of the solve by accounting for balance of multiple entity types – eliminate spikes

Input:

- Priority list of mesh entity types to be balanced (region, face, edge, vertex)
- Partitioned mesh with communication, computation and migration weights for each entity

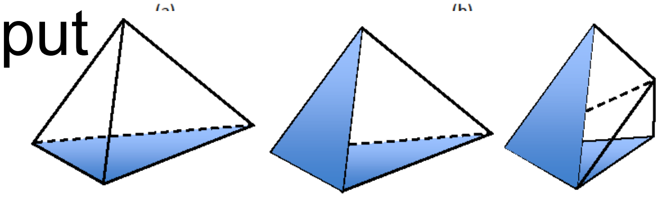
Algorithm:

- From high to low priority if separated by ‘>’ (different groups)
  - From low to high dimension entity types if separated by ‘=’ (same group)
    - *Compute migration schedule (Collective)*
    - *Select regions for migration (Embarrassingly Parallel)*
    - *Migrate selected regions (Collective)*




Ex) “*Rgn>Face=Edge>Vtx*” is the user’s input

**Step 1**: improve balance for mesh regions
**Step 2.1**: improve balance for mesh edges
**Step 2.2**: improve balance for mesh faces
**Step 3**: improve balance for mesh vertices



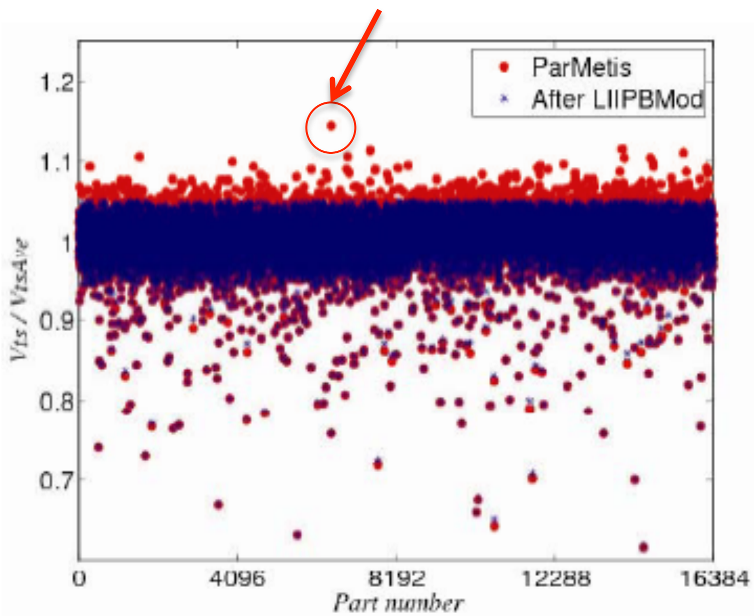
**Mesh element selection**

# ParMA Application Partition Improvement

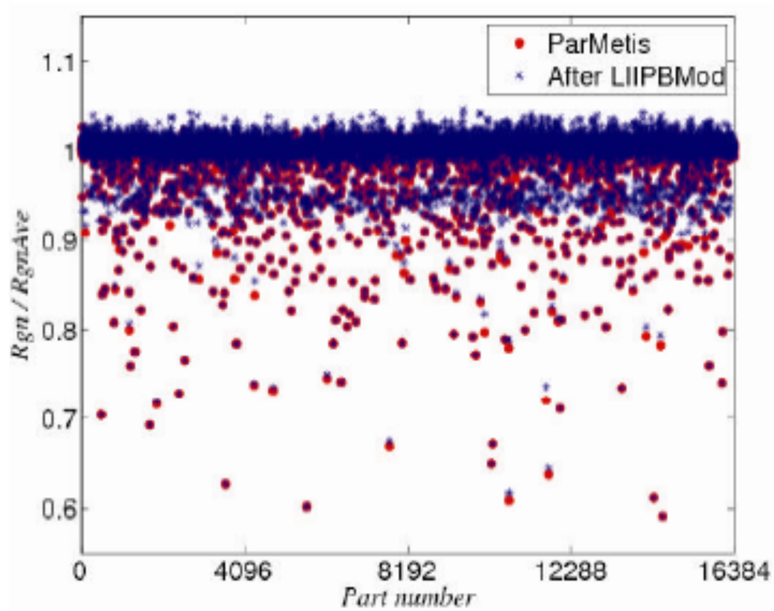
Example of $C^0$, linear shape function finite elements

- Assembly sensitive to mesh element imbalances
- Sensitive to vertex imbalances they hold the dof
  - Heaviest loaded part dictates solver performance
- Element-based partitioning results in spikes of dofs
- Diffusive application of ParMA knocks spikes down – common for 10% increase in strong scaling

***dof*** **imbalance reduced from 14.7% to 4.92%**

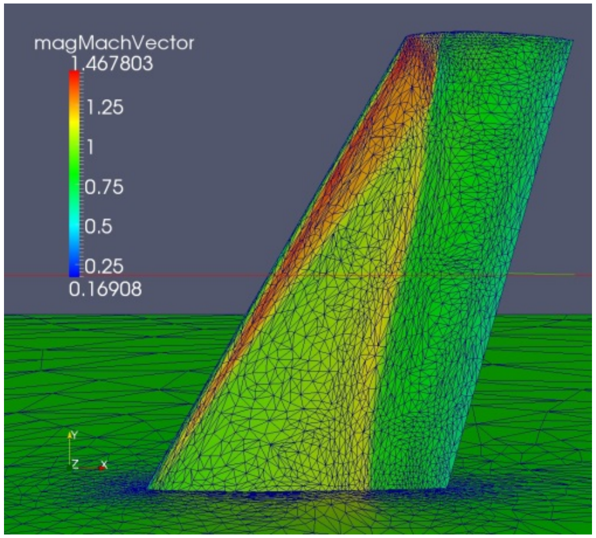


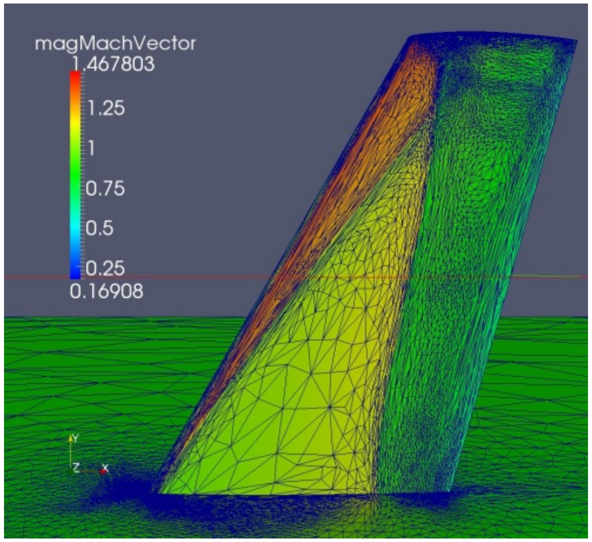

***element*** **imbalance increased from 2.64% to 4.54%**

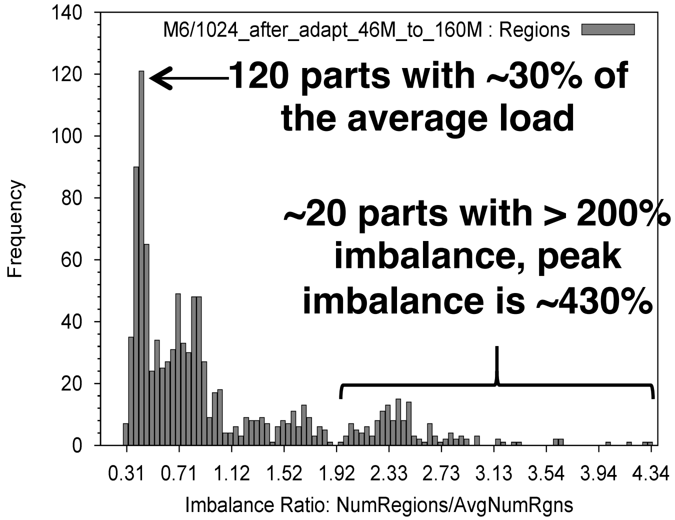


# Predictive Load Balancing

- Mesh modification before load balancing can lead to memory problems - common to see 400% increase on some parts
- Employ predictive load balancing to avoid the problem
  - Assign weights based on what will be refined/coarsened
  - Apply dynamic load balancing using those weights
  - Perform mesh modifications
  - May want to do some local migration




**Histogram of element imbalance in 1024 part adapted mesh on Onera M6 wing if no load balancing is applied prior to adaptation.**

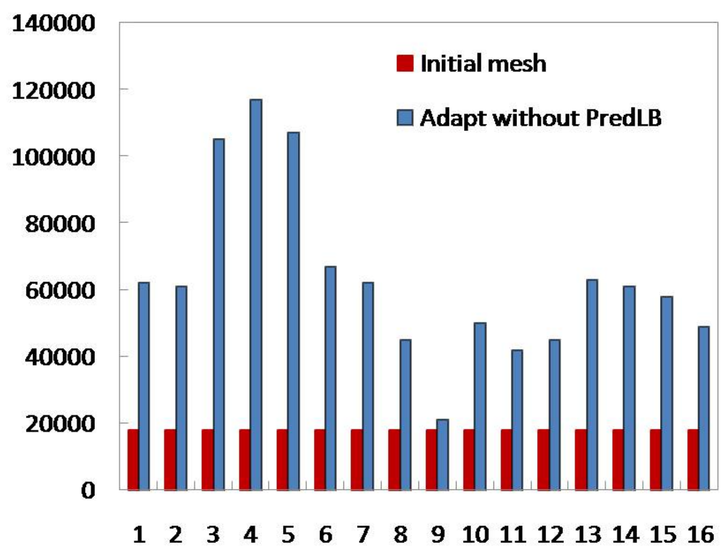
# Predictive Load Balancing

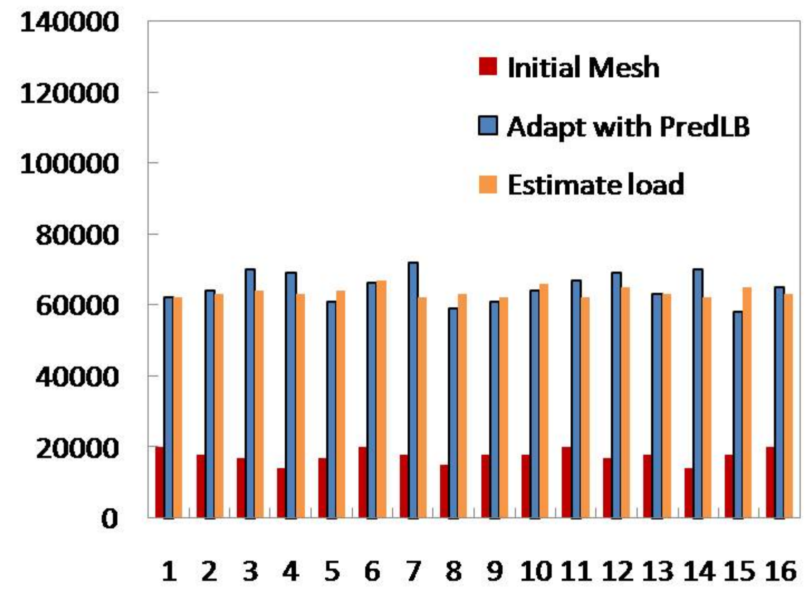
## Algorithm

- Mesh metric field at any point P is decomposed into three orthogonal direction *($e_1$,$e_2$,$e_3$)* and desired length *($h_1$,$h_2$,$h_3$)* in each corresponding direction.
- The volume of desired element (tetrahedron) : $h_1h_2h_3/6$
- Estimate number of elements to be generated:

$$num = \frac{R\_volume}{\sum_{p=1}^{n_{en}} h_1(p)h_2(p)h_3(p)/6n_{en}}$$

- "num" is scaled to a good range before it is specified as a weight to graph nodes



# General Mesh Modification for Mesh Adaptation

Goal is the flexibility of remeshing with added advantages

- Supports general changes in mesh size including anisotropy
- Can deal with any level of geometric domain complexity
- Can obtain level of accuracy desired
- Solution transfer can be applied incrementally

Given the “mesh size field”:

- Drive the mesh modification loop at the element level
  - Look at element edge lengths and shape (in transformed space)
  - If not satisfied select “best” modification
  - Elements with edges that are too long must have edges split or swapped out
  - Short edges eliminated
- Continue until size and shape is satisfied or no more improvement possible

Determination of “best” mesh modification

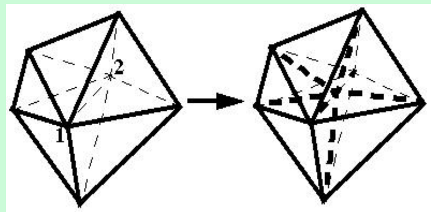
- Selection of mesh modifications based on satisfaction of the element requirements
- Appropriate considerations of neighboring elements
- Choosing the “best” mesh modification

# Mesh Adaptation by Local Mesh Modification

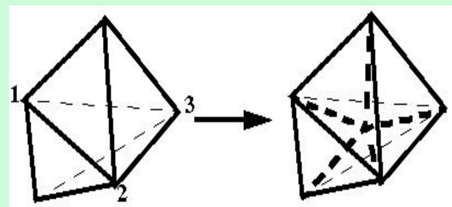
Controlled application of mesh modification operations including dealing with curved geometries, anisotropic meshes

Base operators

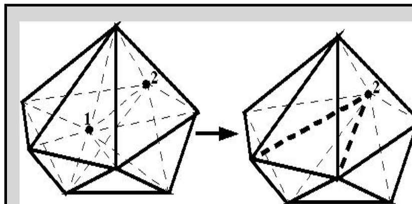
- swap
- collapse
- Split
- move



Edge split



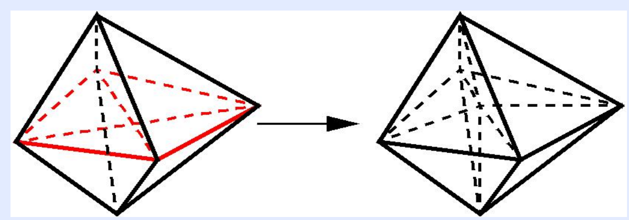
face split



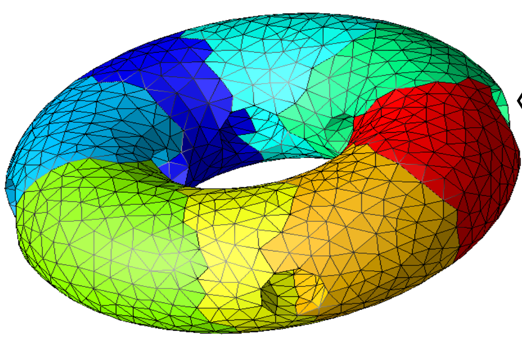
Edge collapse

Compound operators chain single step operators

- Double split collapse operator
- Swap(s) followed by collapse operator
- Split, then move the created vertex
- Etc.

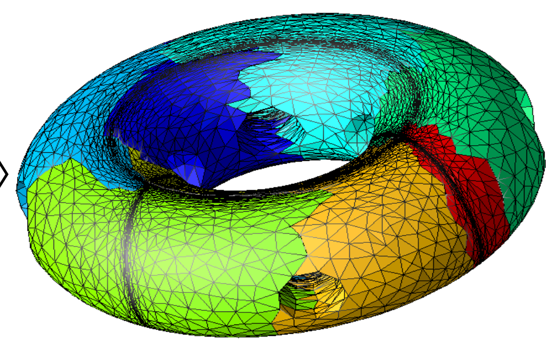


Double split collapse to remove the red sliver



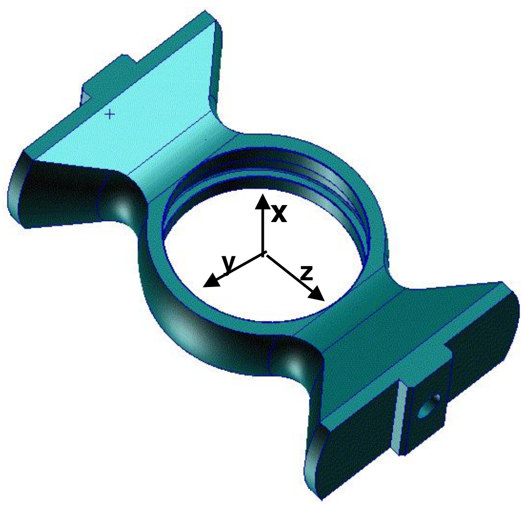
initial mesh (20,067 tets)

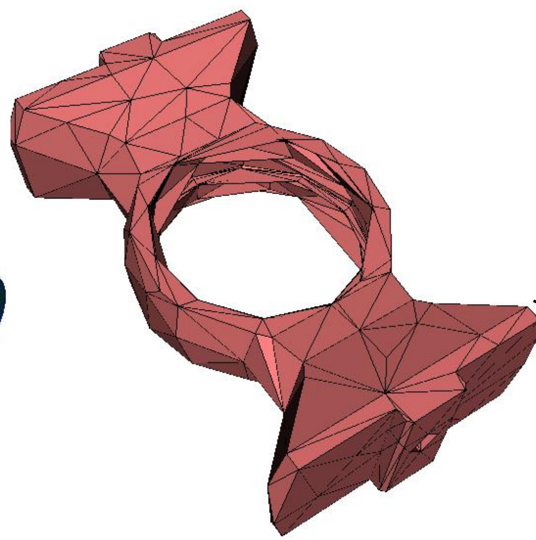
adapted mesh (approx. 2M tets)

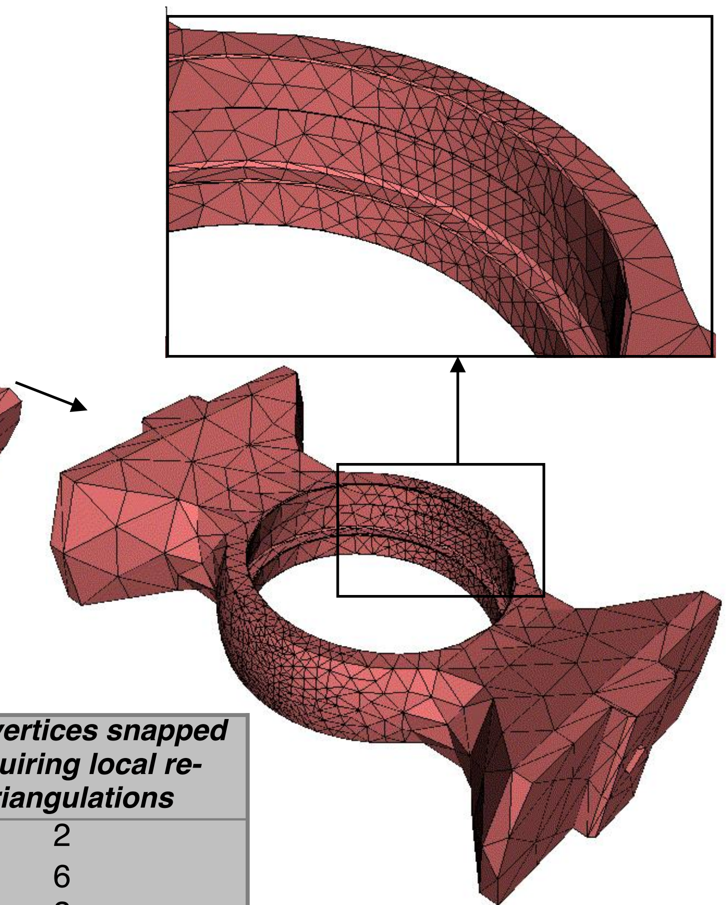


# Accounting for Curved Domains During Refinement

- Moving refinement vertices to boundary required mesh modification (see IJNME paper, vol58 pp247-276, 2003 )
- Coarse initial mesh and the mesh after multiple refinement/coarsening



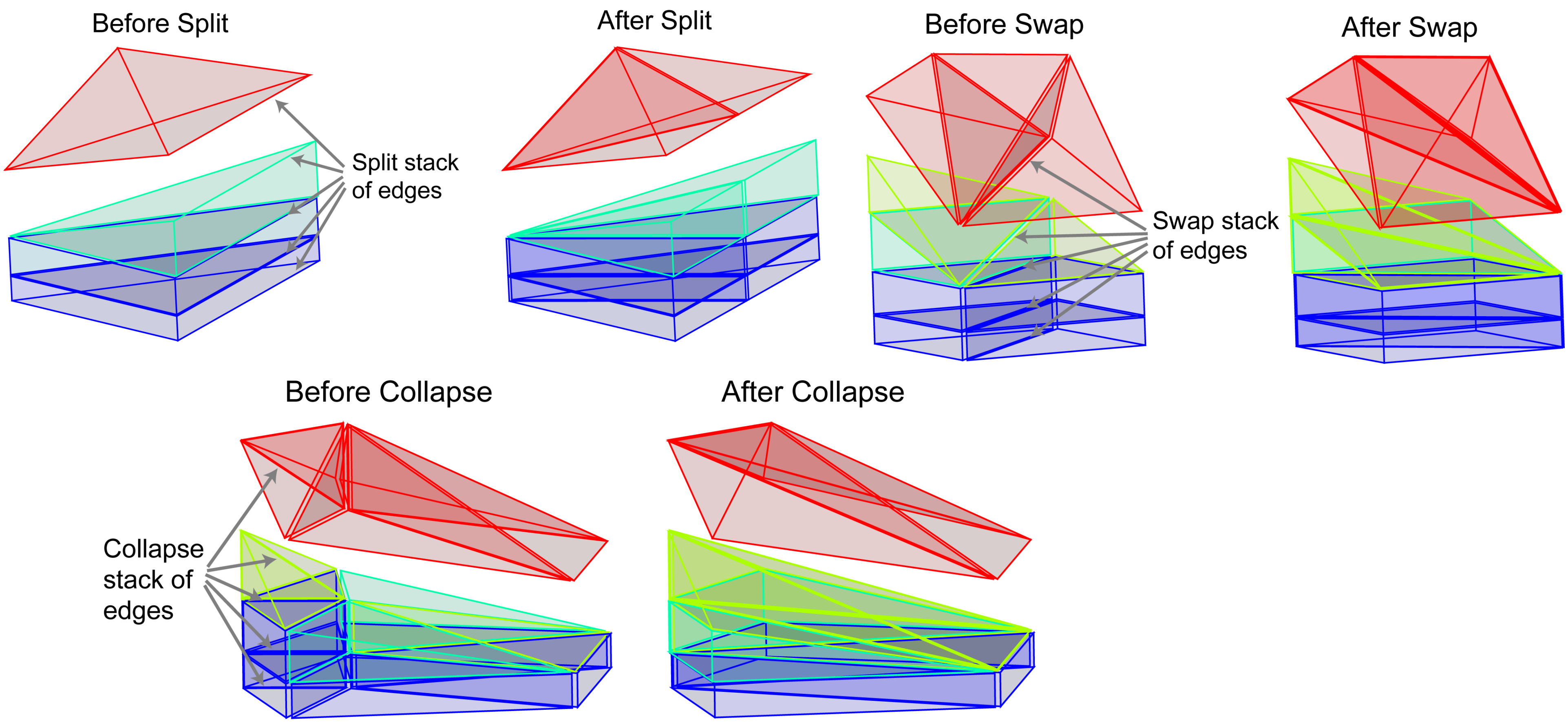


- Operations to move refinement vertices

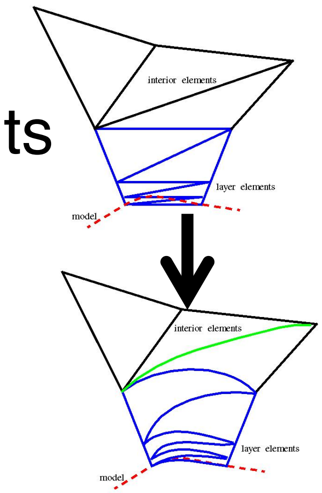
| ***Iterations of adaptations*** | ***# of vertices to be snapped*** | ***# of vertices snapped by a reposition*** | ***# of vertices snapped by local modifications*** | ***# of vertices snapped requiring local re-triangulations*** |
|---|---|---|---|---|
| **1** | 342 | 204 | 136 | 2 |
| **2** | 485 | 369 | 110 | 6 |
| **3** | 340 | 286 | 52 | 2 |
| **4** | 74 | 34 | 40 | - |
| **5** | 26 | 3 | 23 | - |

# Matching Boundary Layer and Interior Mesh

A modification operation on any layer is propagated through the stack and interface elements to preserve attributes of layered elements.



# Curved Elements for Higher-Order Methods

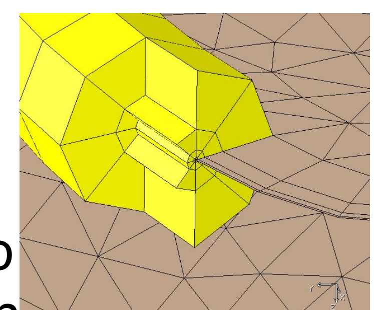



Requirements

- Coarse, strongly graded meshes with curved elements
- Must ensure the validity of curved elements
- Shape measure for curved elements $Q = Q_s \times Q_c$

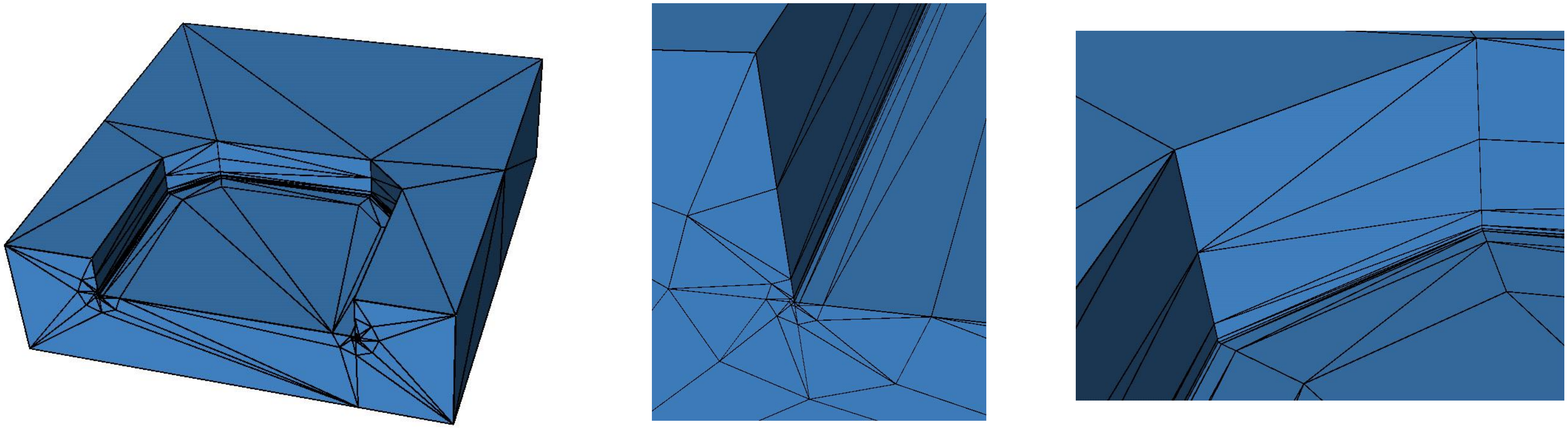
  $Q_s$ - standard straight sided measure in 0-1 format

  $Q_c$ - 0-1 curved measure (det. of Jacobian variation)
- Element geometric order and level of geometric approximation need to be related to geometric shape order
- Steps in the procedure (for optimum convergence rate)
  - Automatic identification and linear mesh at singular features
  - Generate coarse surface mesh accounting for the boundary layers
  - Curve coarse surface mesh to boundary
  - Curve graded linear feature isolation mesh
  - Generate coarse linear interior mesh
  - Modify interior linear mesh to ensure validity with respect to the curved surface and graded linear feature isolation mesh

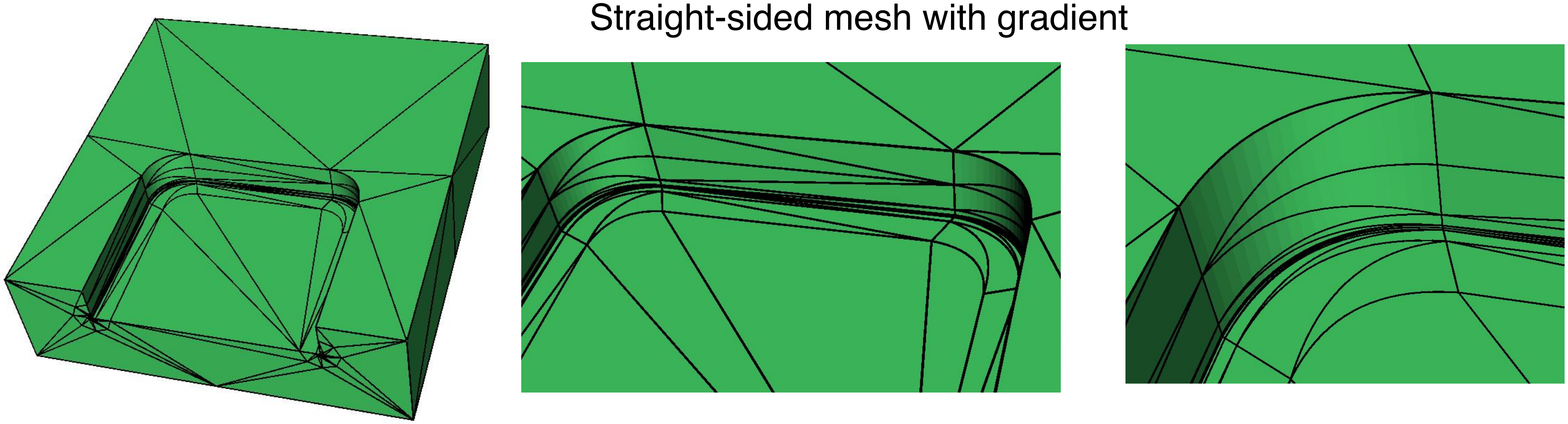


# Example p-Version Mesh

- Isolation on model edges



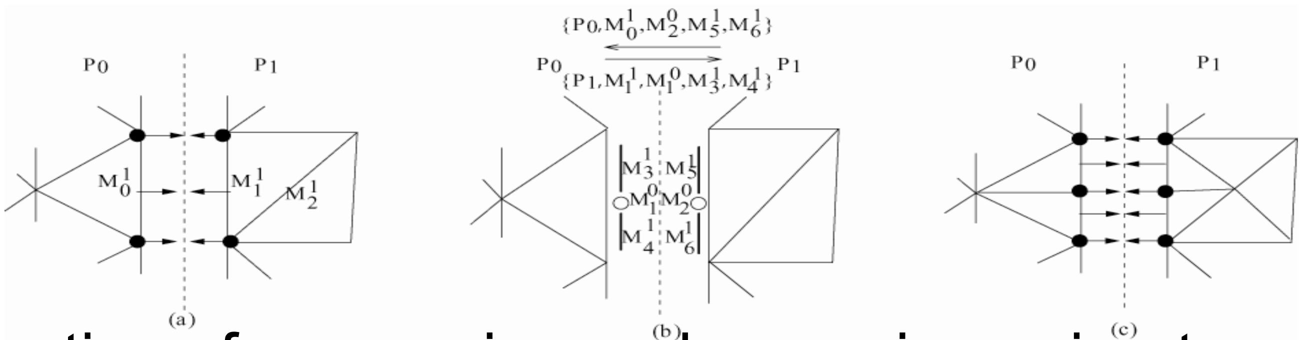
Straight-sided mesh with gradient



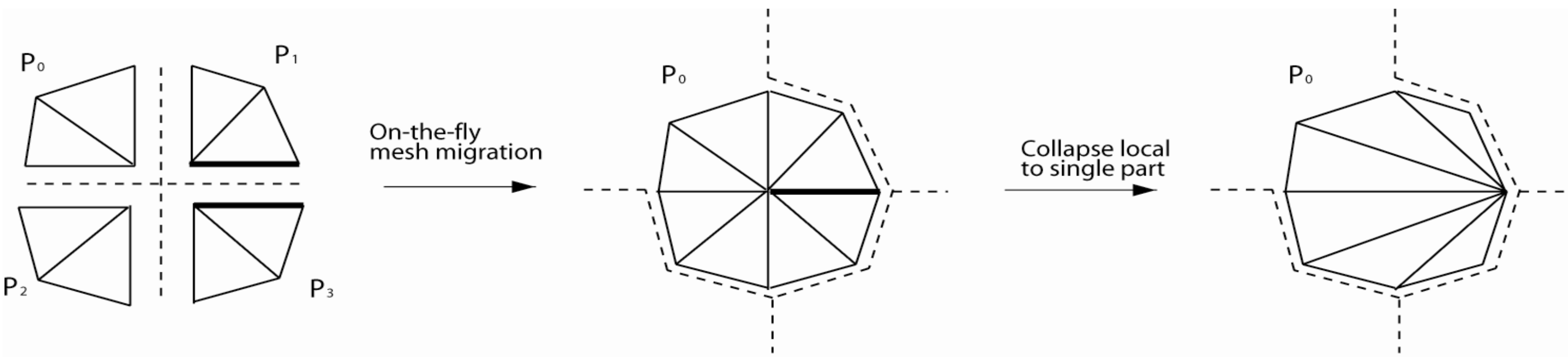
Curved mesh with gradient

# Parallel Mesh Adaptation

Parallelization of refinement: perform on each part and synchronize at inter-part boundaries.




Parallelization of coarsening and swapping: migrate cavity (on-the-fly) and perform operation locally on one part.

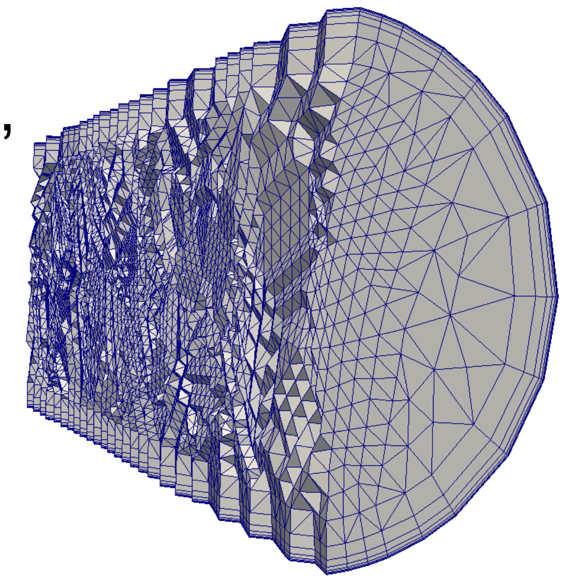


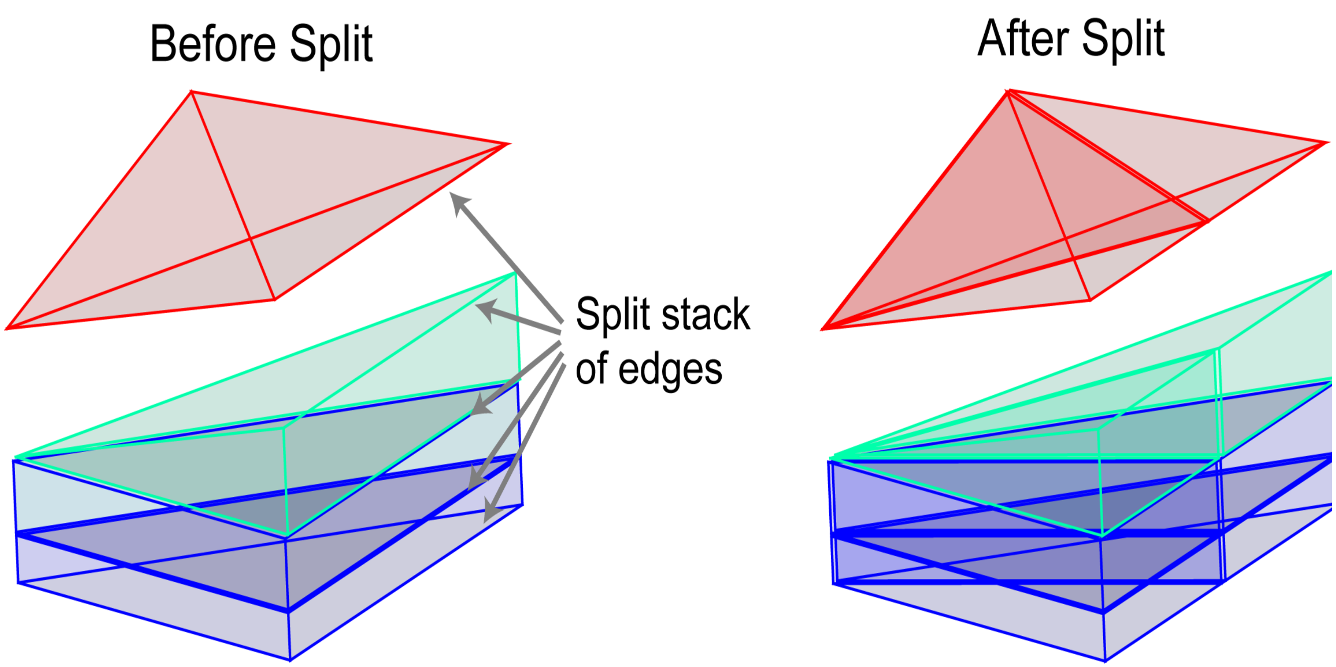

Support for parallel mesh modification requires update of evolving communication-links between parts and dynamic mesh partitioning.

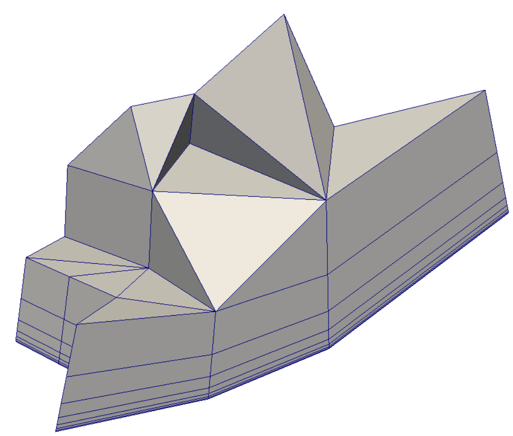
# Boundary Layer Mesh Adaptation

## Boundary Layer stacks in P-sets

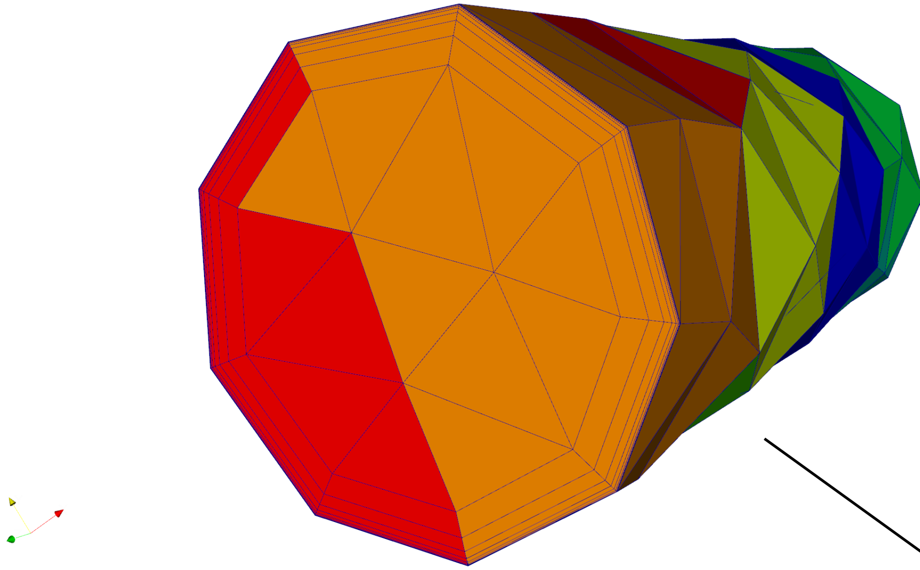
- Mesh entities contained in a set are unique, and are not part of the boundary of any higher dimension mesh entities
- Migrate a set and constituting entities to another part together





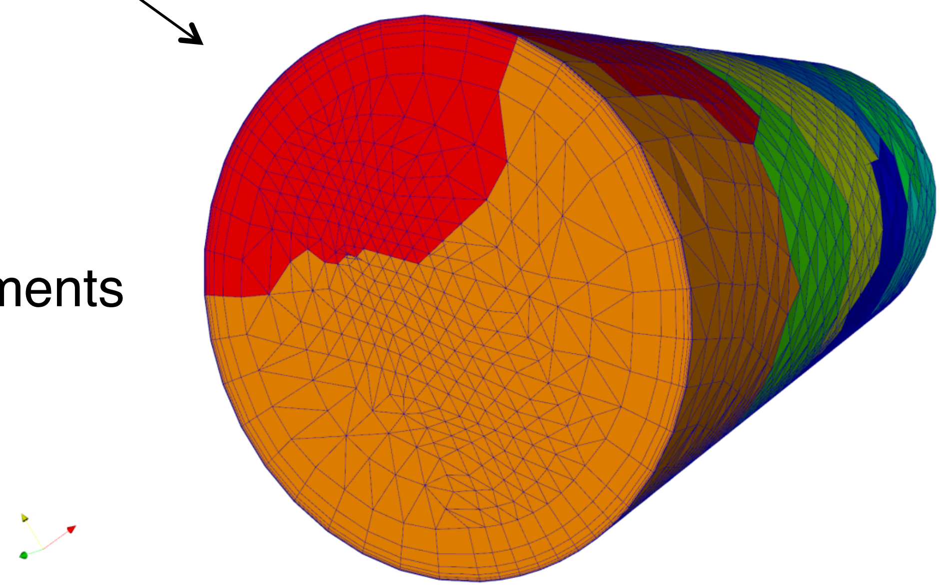



# Parallel Boundary Layer Adaptation



Initial mesh of 2k elements

Refinement and node repositioning with limited coarsening and swapping



Final mesh of 210k elements

# Mesh Adaptation to an Anisotropic Mesh Size Field

Define desired element size and shape distribution following mesh metric

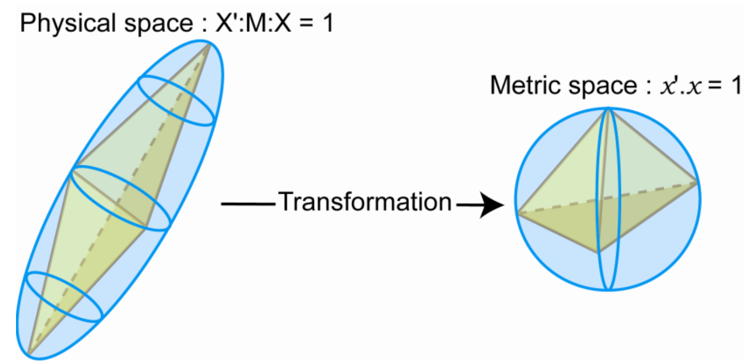
Transformation matrix field T(x,y,z)

$$T(x,y,z) = \underbrace{\begin{bmatrix} 1/h_1 & 0 & 0 \\ 0 & 1/h_2 & 0 \\ 0 & 0 & 1/h_3 \end{bmatrix}}_{\text{Distortion}} \cdot \underbrace{\begin{bmatrix} \vec{e}_1 \\ \vec{e}_2 \\ \vec{e}_3 \end{bmatrix}}_{\text{Rotation}}$$

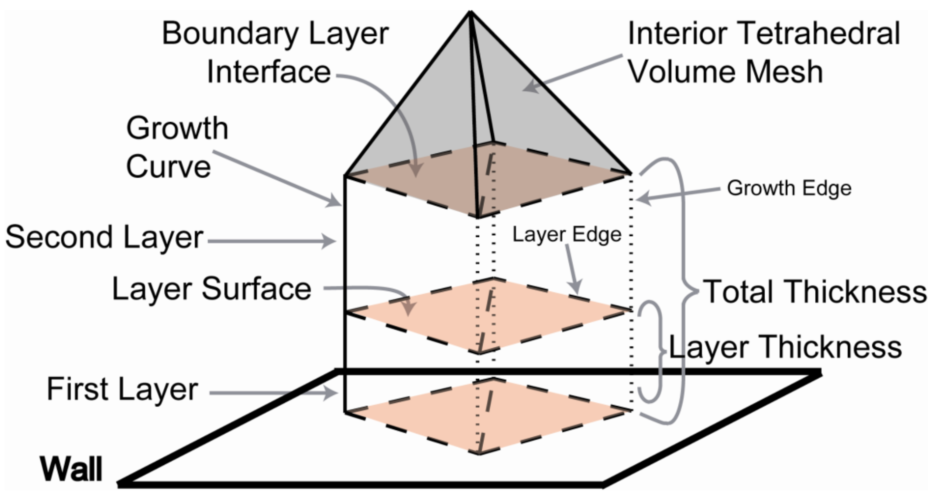
$\vec{e}_1, \vec{e}_2, \vec{e}_3$ : Unit vectors associated with three principle directions

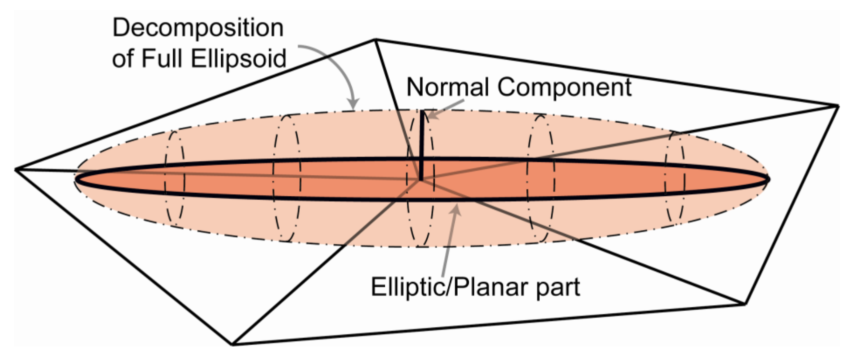
$h_1, h_2, h_3$ : Desired mesh edge lengths in these directions

Ellipsoidal in physical space transformed to normalized sphere

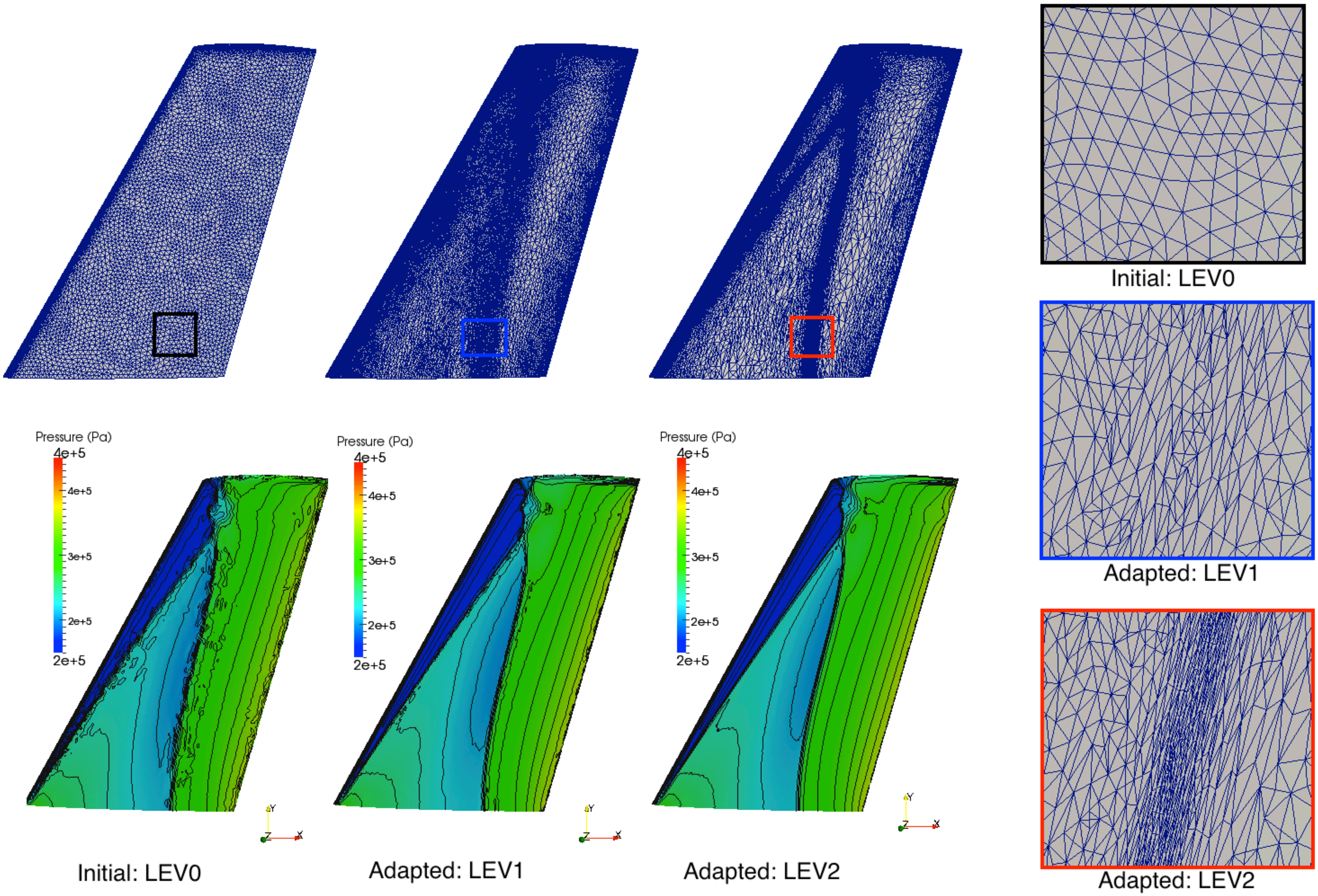



Decomposition of boundary layers into layer surfaces (2D) and a thickness (1D) mesh
In-plane adaptation uses projected Hessian, thickness adaptation based on BL theory

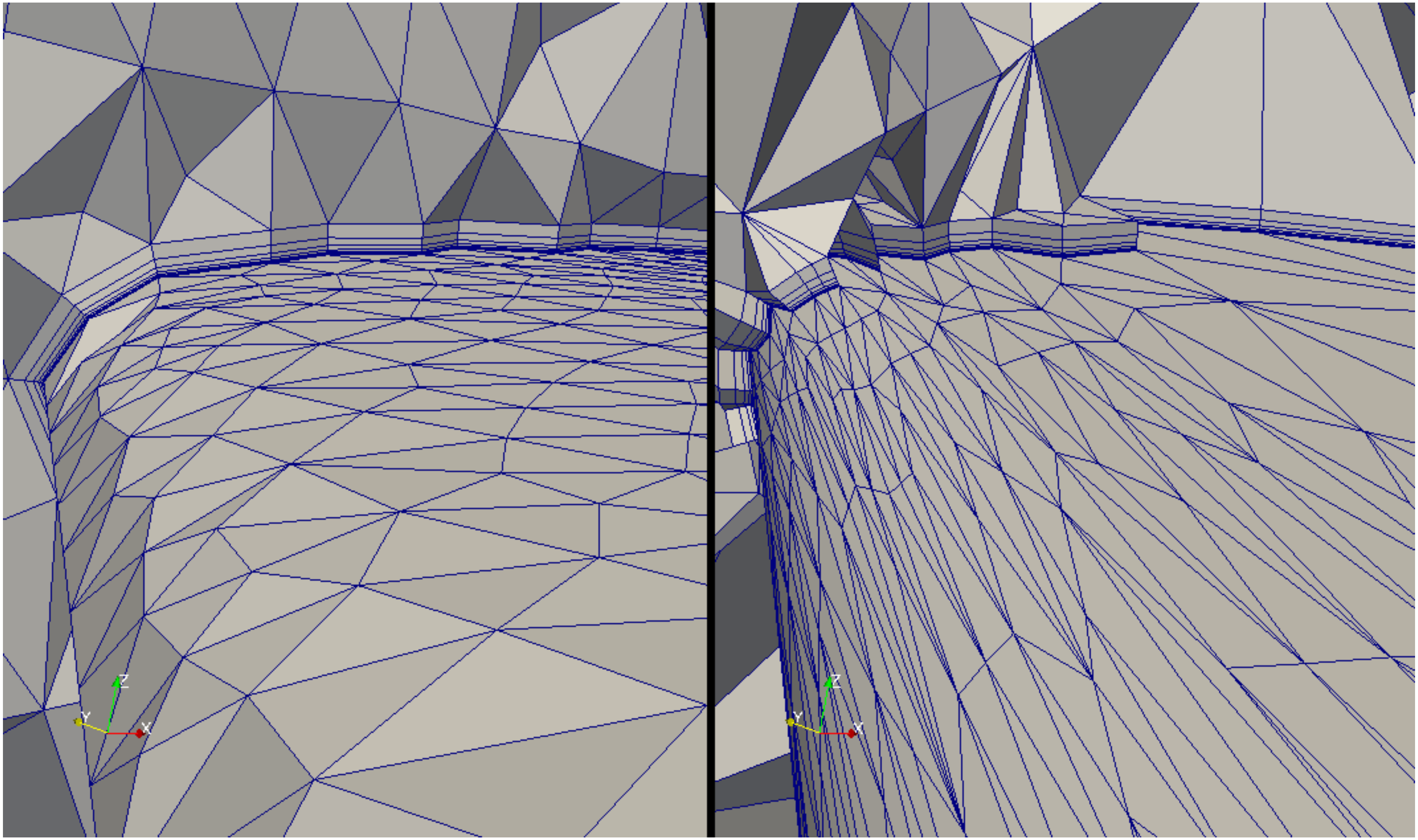


# Example 2 – M6 Wing

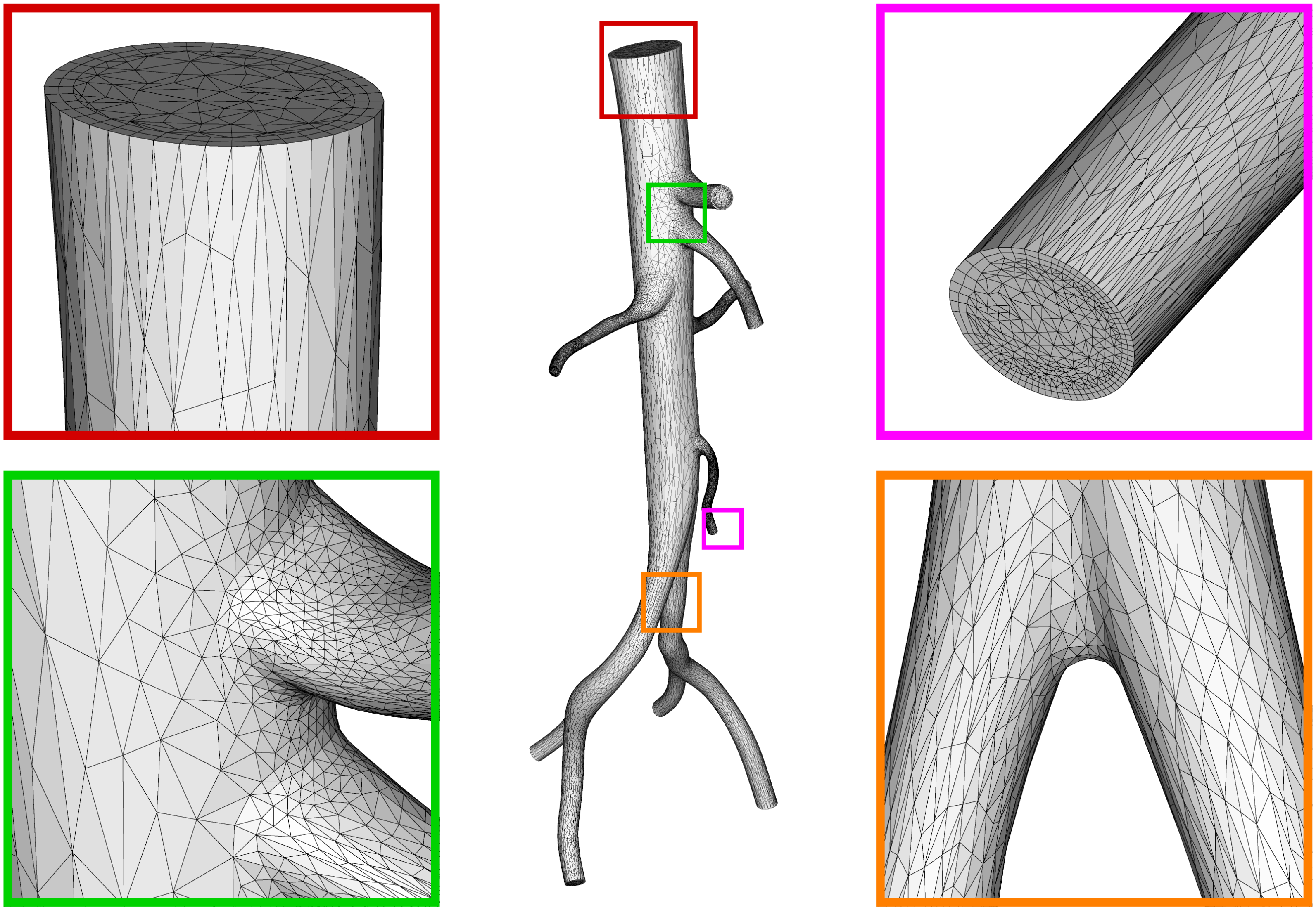


# Example of Anisotropic Adaptation



# Example

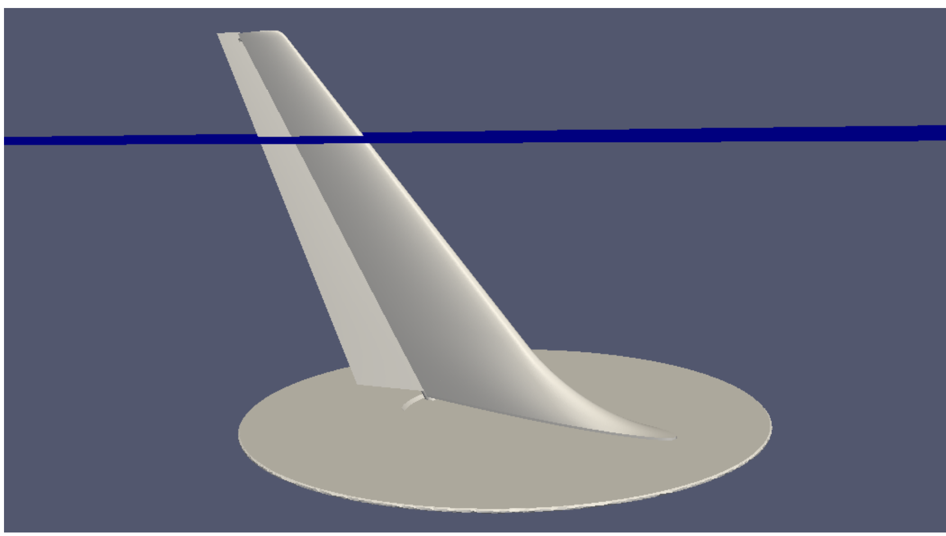
Surface of adapted mesh for human abdominal aorta



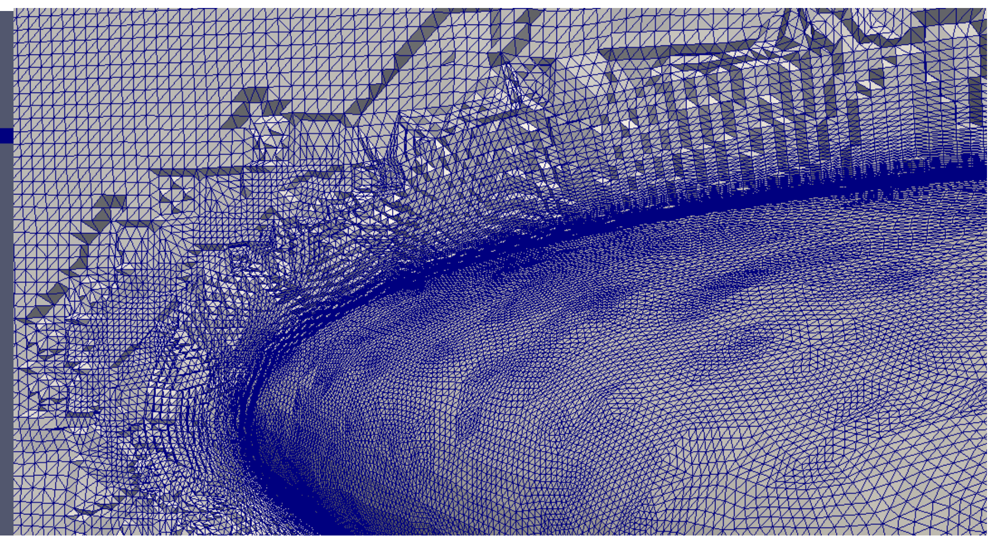
# Component-Based Construction of Adaptive Loops

Building on the unstructured mesh infrastructure

- Employs a component-based approach interacting through functional interfaces
- Being used to construct parallel adaptive loops for codes
- Recently used for a 92B element mesh on ¾ million cores

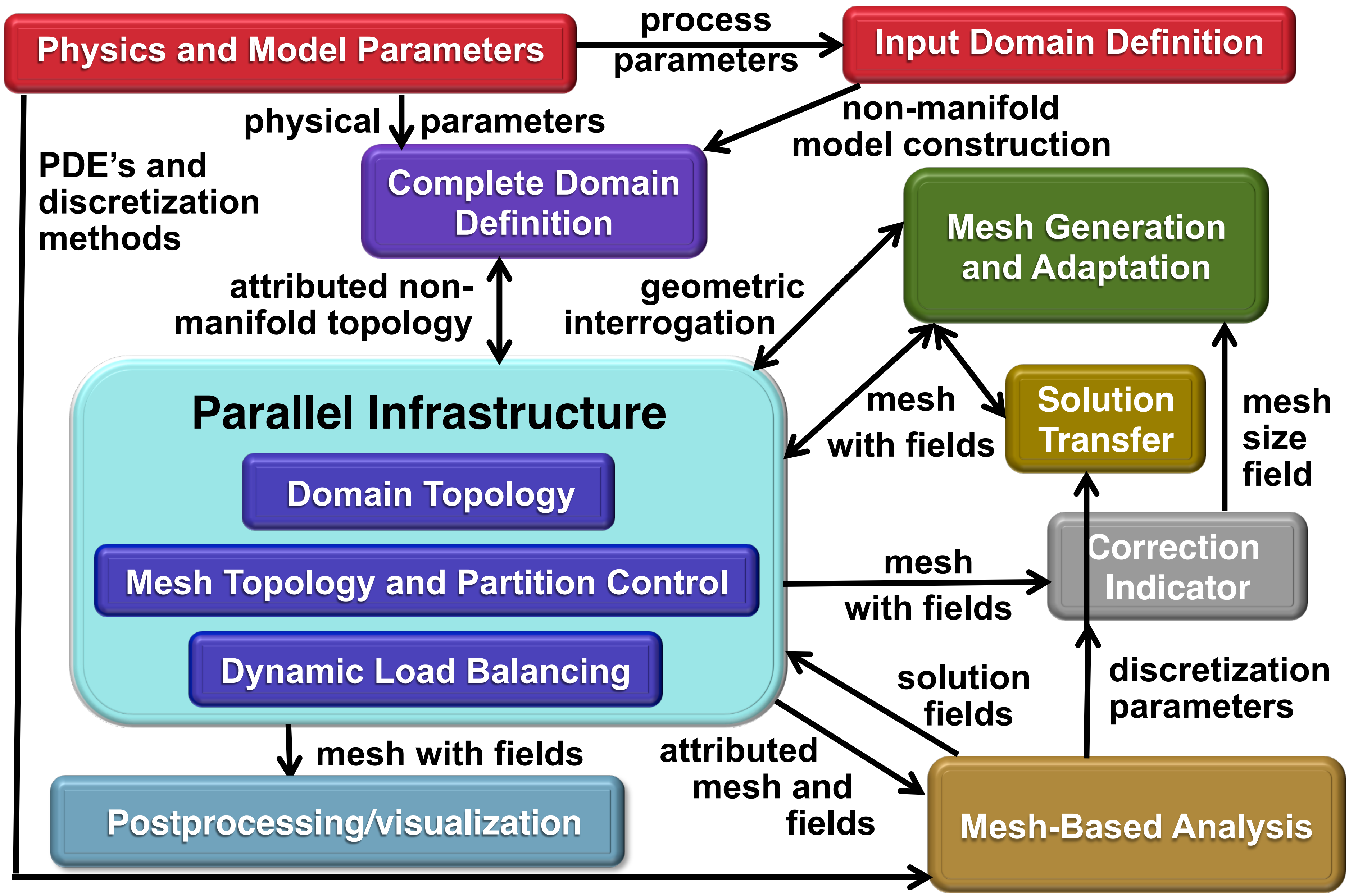


Overall geometry and slice plane shown



11B element mesh

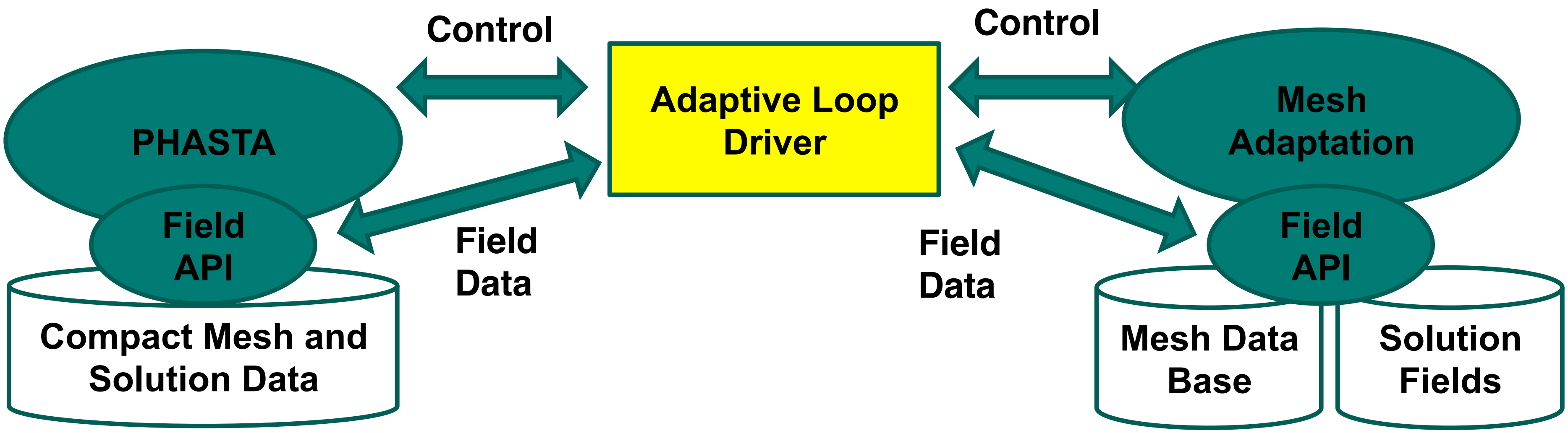
# Component-Based Unstructured Mesh Infrastructure



# In-Memory Adaptive Loop

Mapping data between component data structures and executing memory management

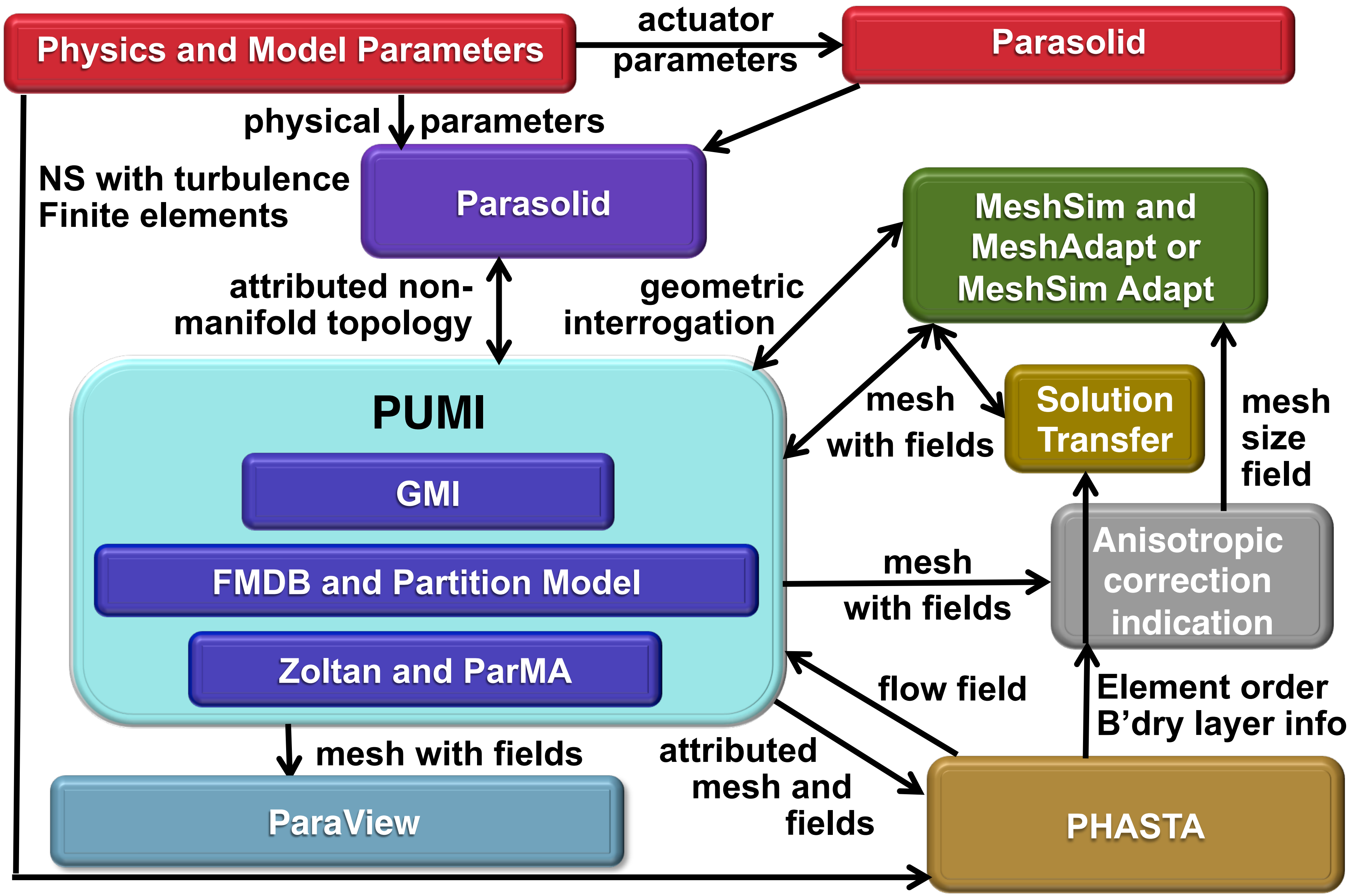
- Component integrated using functional interfaces
- Change/Add components with minimal development costs




Comparison of file-based and in-memory transfer for PHASTA

- 85M element mesh on Hopper
- On 512 cores file based took 49 sec and in-memory 2 sec
- On 2048 cores file based took 91 sec and in-memory 1 sec

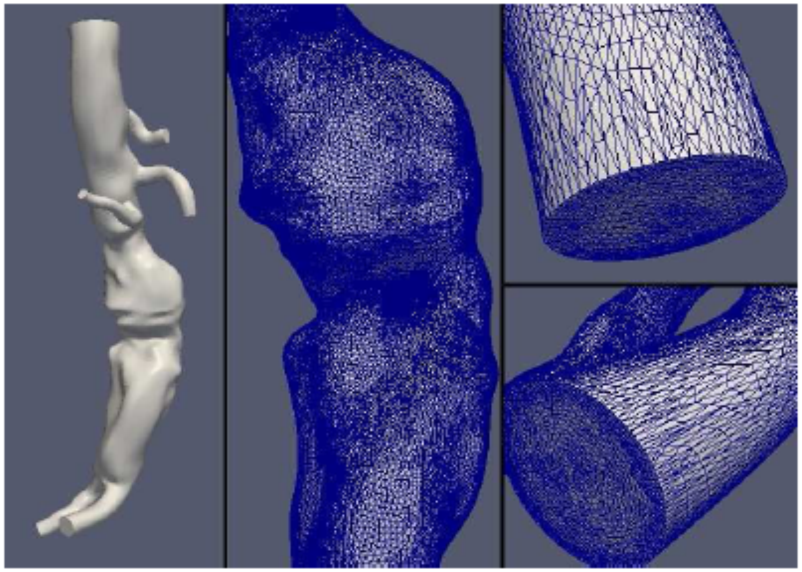
# Active Flow Control Simulations



# Example of Scalable Solver: PHASTA

Excellent strong scaling

- Implicit time integration
- Employs the partitioned mesh for system formulation and solution
- Specific number of ALL-REDUCE communications also required

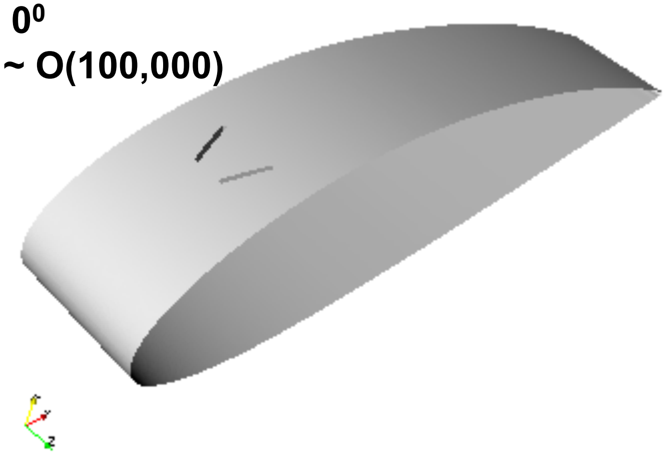


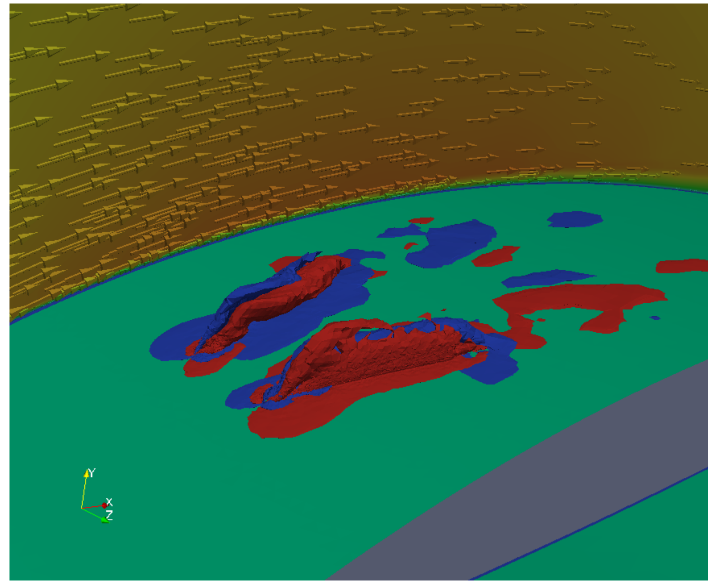
Strong Scaling Results

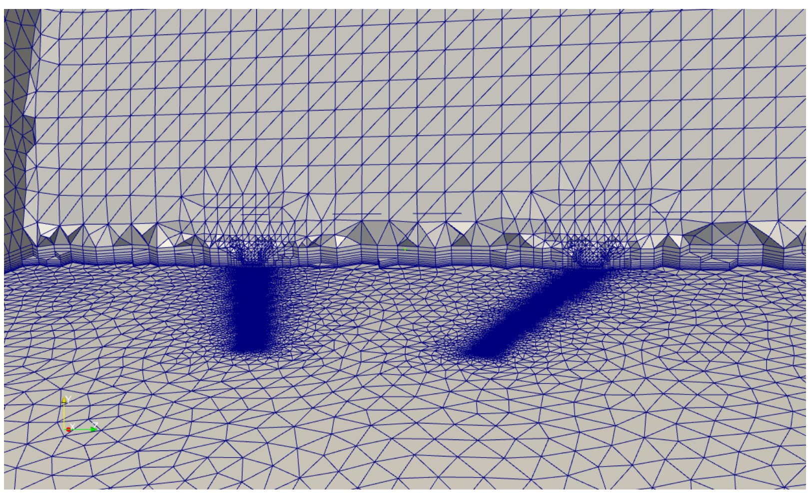
| 1.07$B$ elements mesh | | Intrepid:IBM BG/P | | Kraken:Cray XT5 | | JuGene:IBM BG/P | |
|---|---|---|---|---|---|---|---|
| num. of cores | avg. elem./core | time | s-factor | time | s-factor | time | s-factor |
| 4,096 (base) | 261,600 | 844.38 | 1 | 311.34 | 1 | 845.68 | 1 |
| 8,192 | 130,800 | 427.33 | 0.99 | 144.23 | 1.08 | – | – |
| 16,384 | 65,400 | 217.05 | 0.97 | 73.06 | 1.07 | – | – |
| 32,768 | 32,700 | 109.87 | 0.96 | 39.35 | 0.97 | – | – |
| 65,536 | 16,350 | 58.65 | 0.91 | 28.04 | 0.69 | – | – |
| 98,304 | 10,900 | 39.06 | 0.90 | 18.67 | 0.70 | – | – |
| 131,072 | 8,175 | 29.68 | 0.89 | – | – | – | – |
| 163,840 | 6,540 | 24.12 | 0.88 | – | – | – | – |
| 294,912 | 3,630 | – | – | – | – | 14.39 | 0.82 |

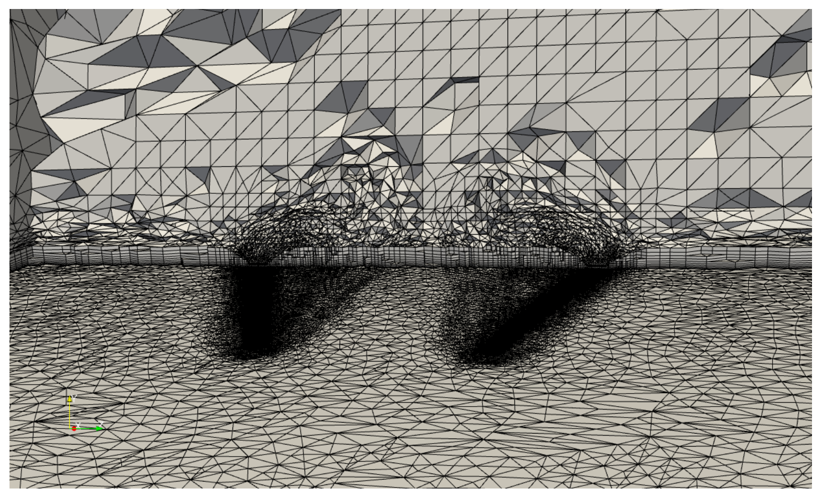
# Mesh Adaptivity for Synthetic Jets (O. Sahni)

$f_{act}$ = 2,300Hz
$\alpha = 0^0$
*Re* ~ O(100,000)

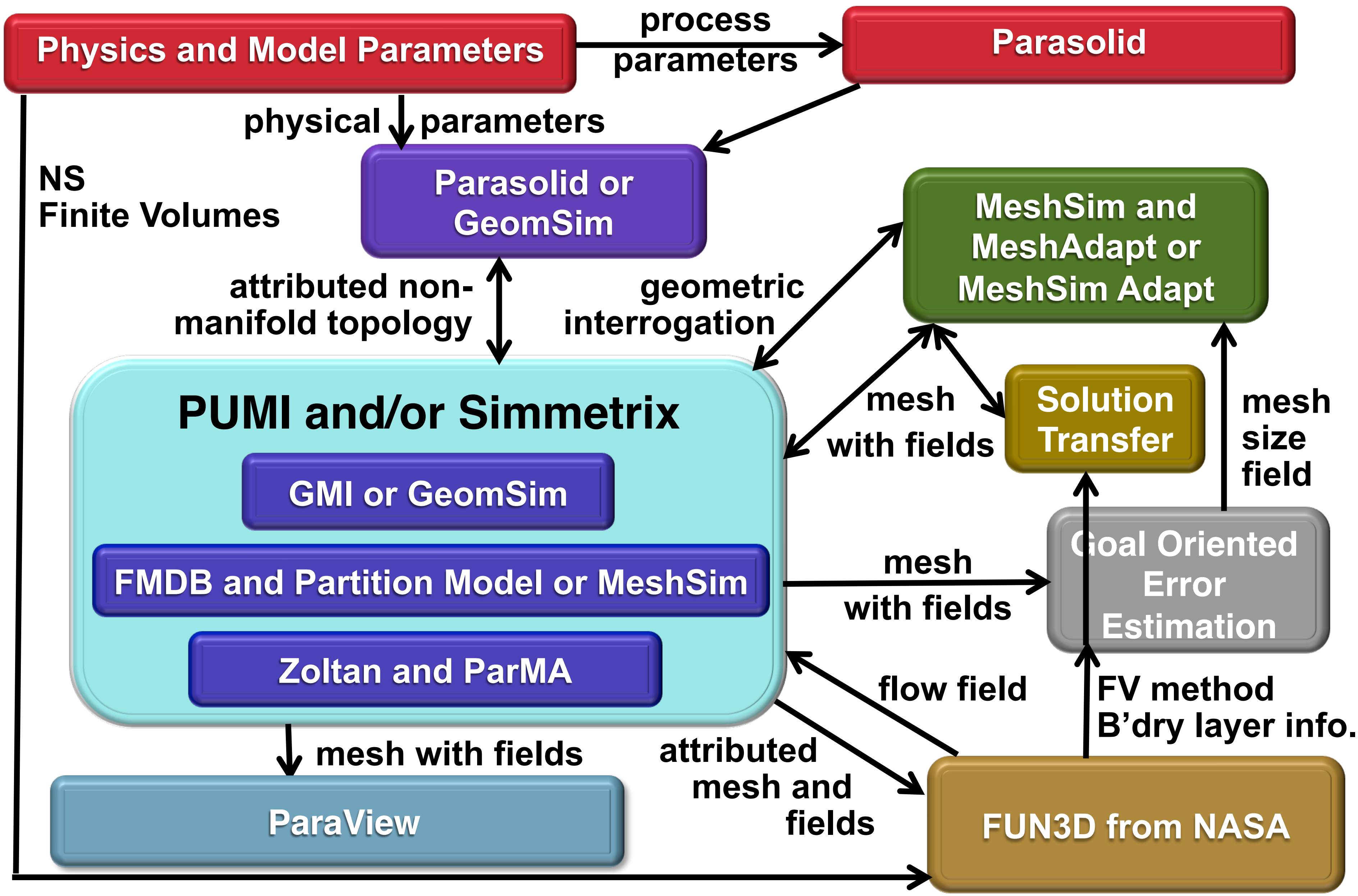






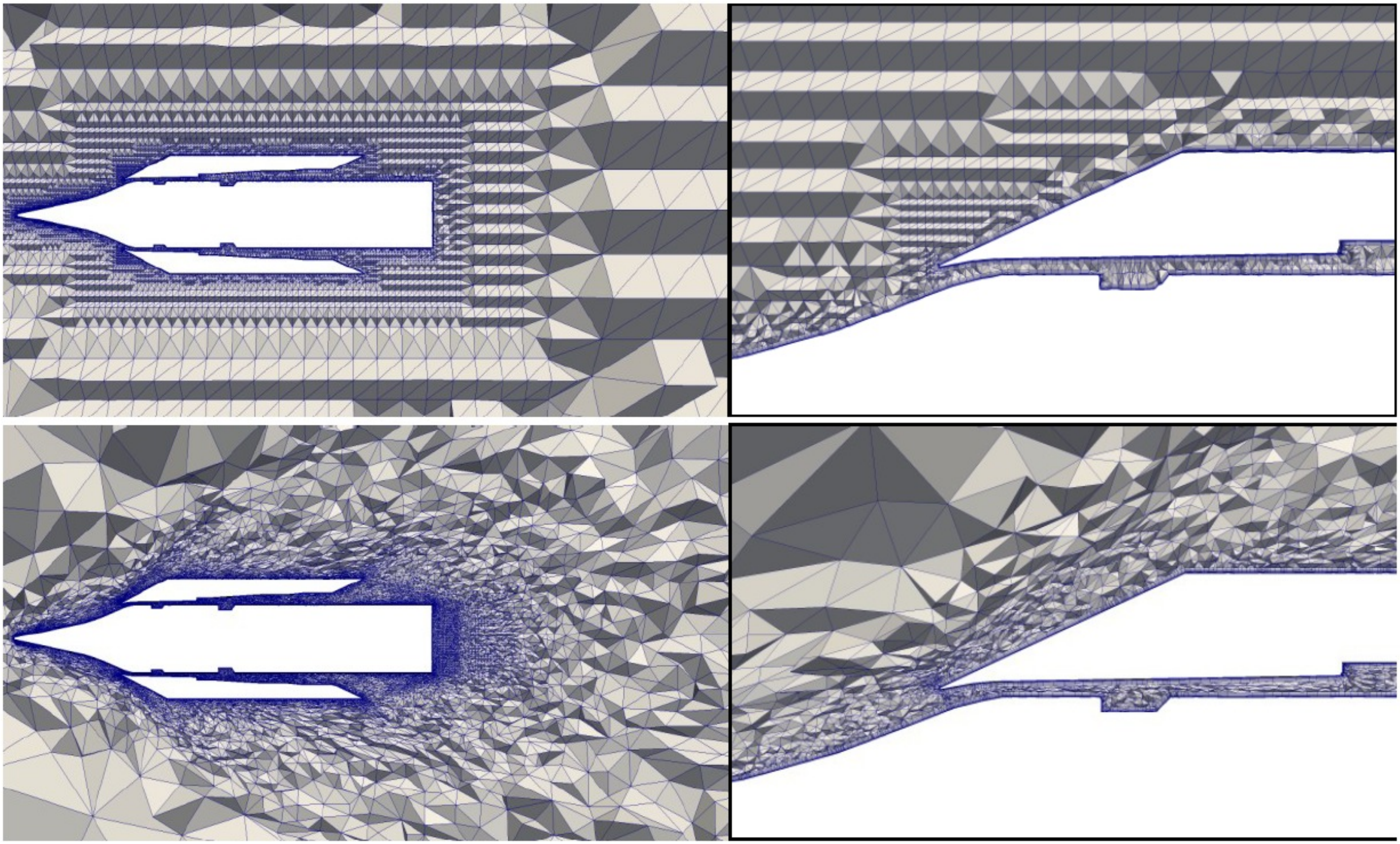


# Aerodynamics Simulations



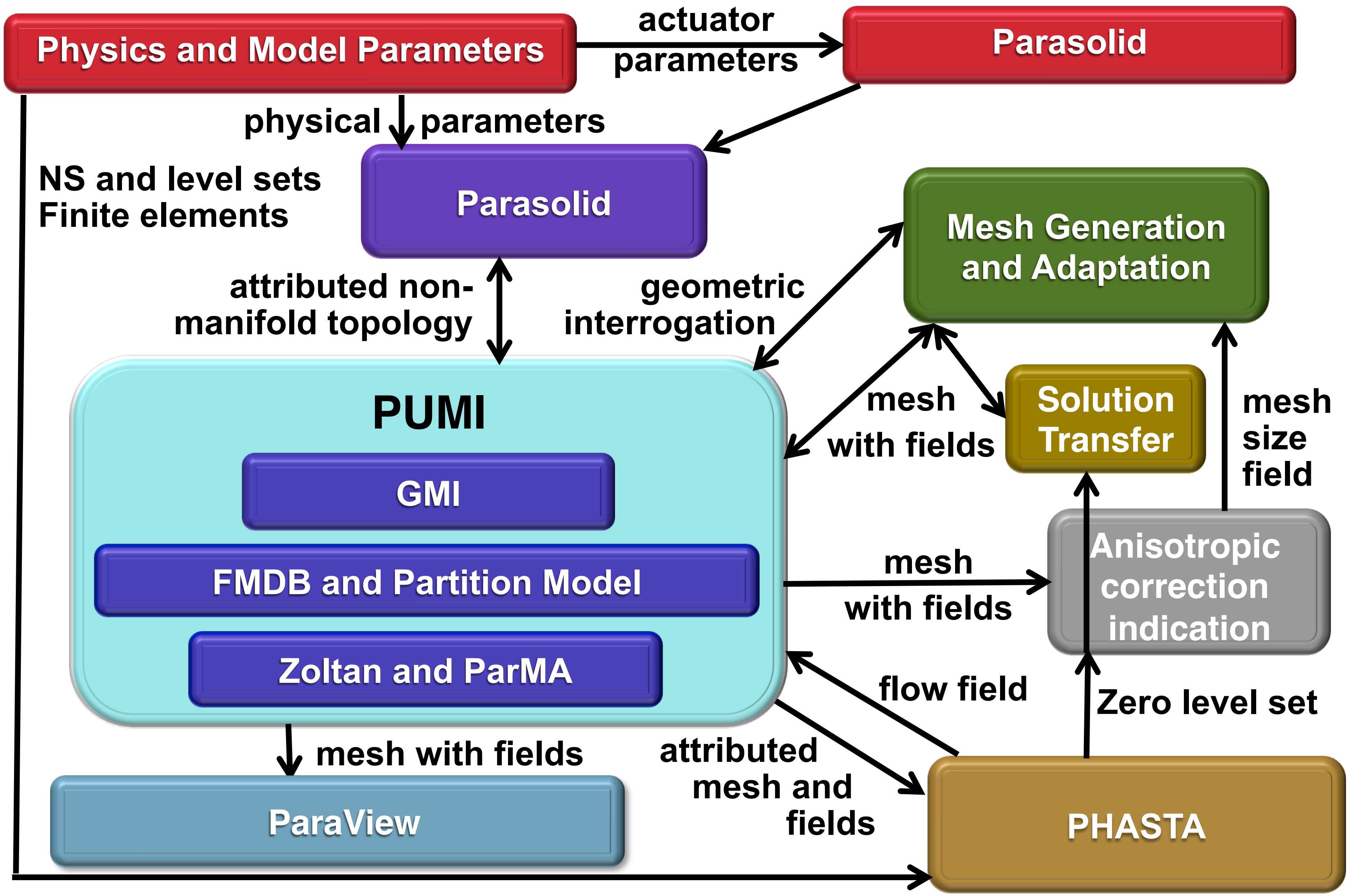
# Application Result - Scramjet Engine

**Initial Mesh**



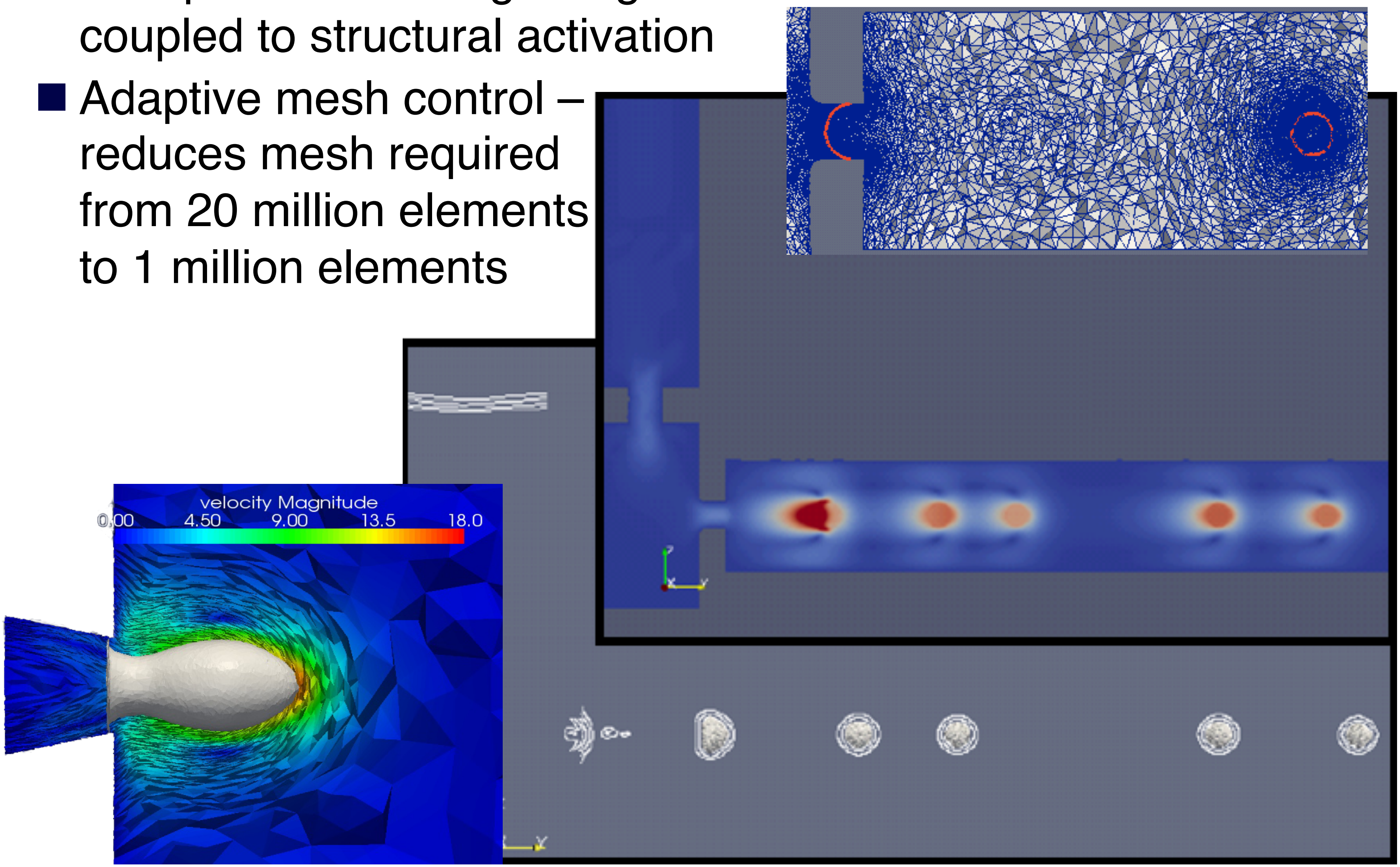
**Adapted Mesh**

# Adaptive Two-Phases Flow

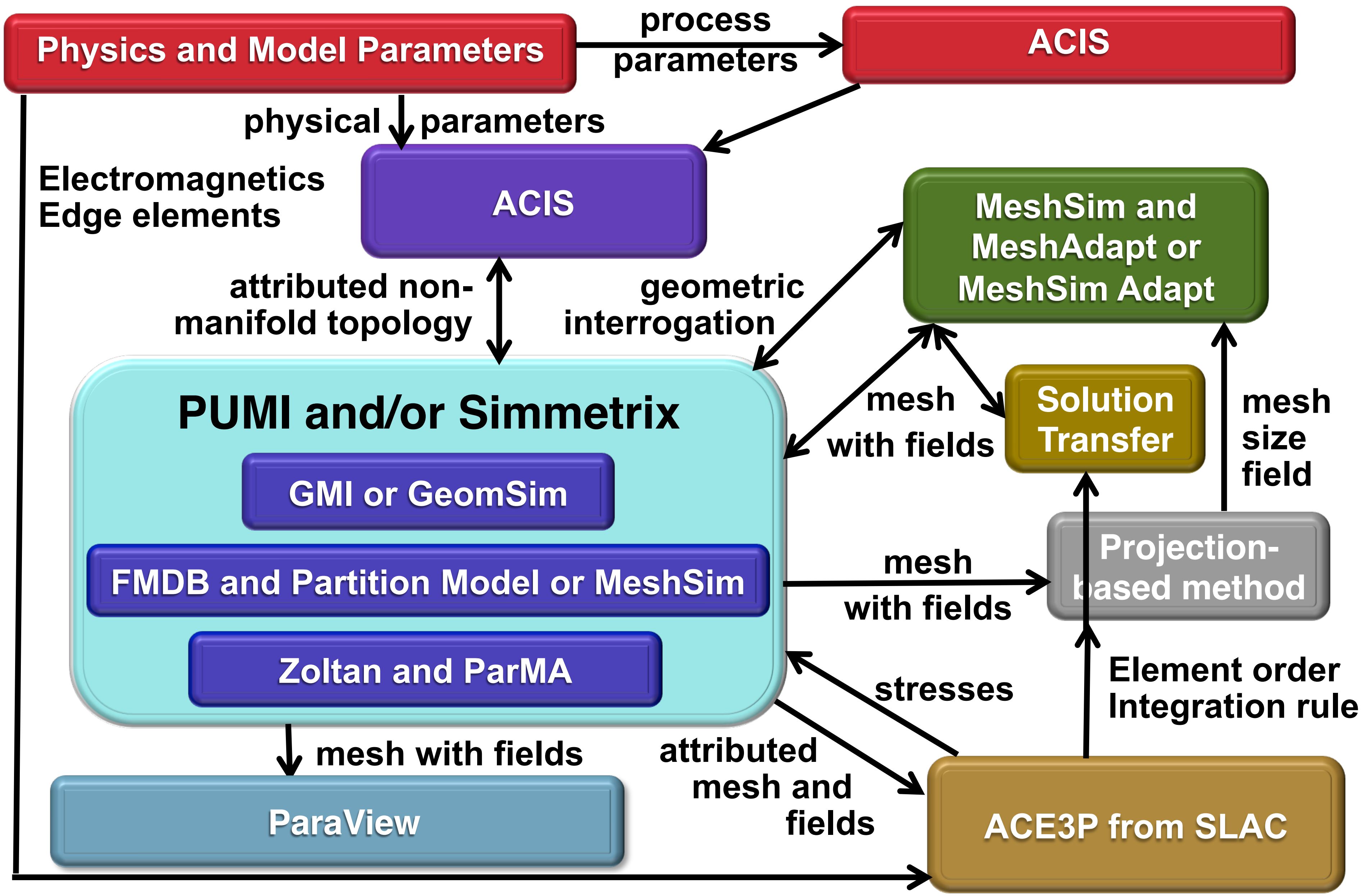


# Adaptive Simulation of Two-Phase Flow

- Two-phase modeling using level-sets coupled to structural activation
- Adaptive mesh control – reduces mesh required from 20 million elements to 1 million elements

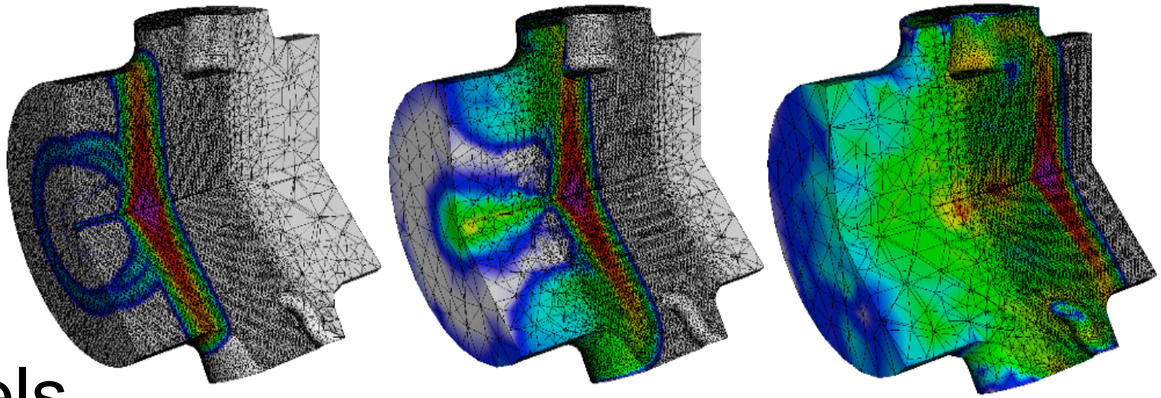


# Electromagnetics Analysis

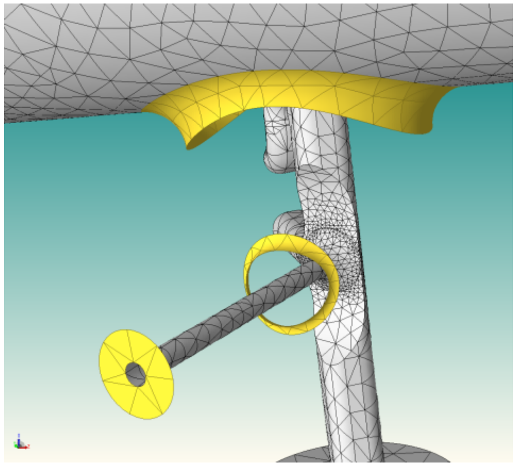


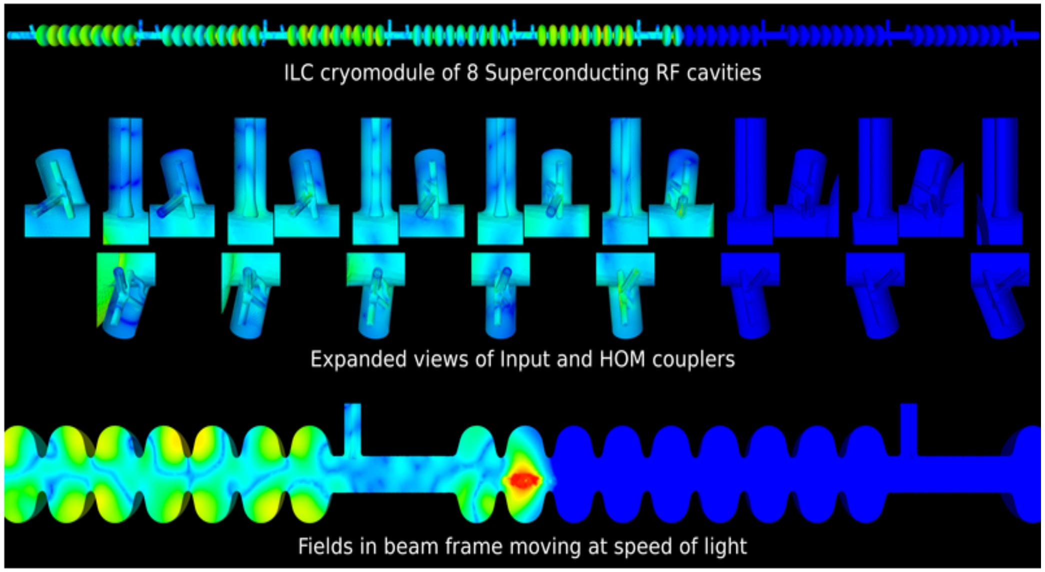
# *Adaptive Control Coupled with PIC Method*

Adaptation based on
- Tracking particles (needs fine mesh)
- Discretization errors



Full accelerator models
- Approaching 100 cavities
- Substantial internal structure
- Meshes with several hundred million elements



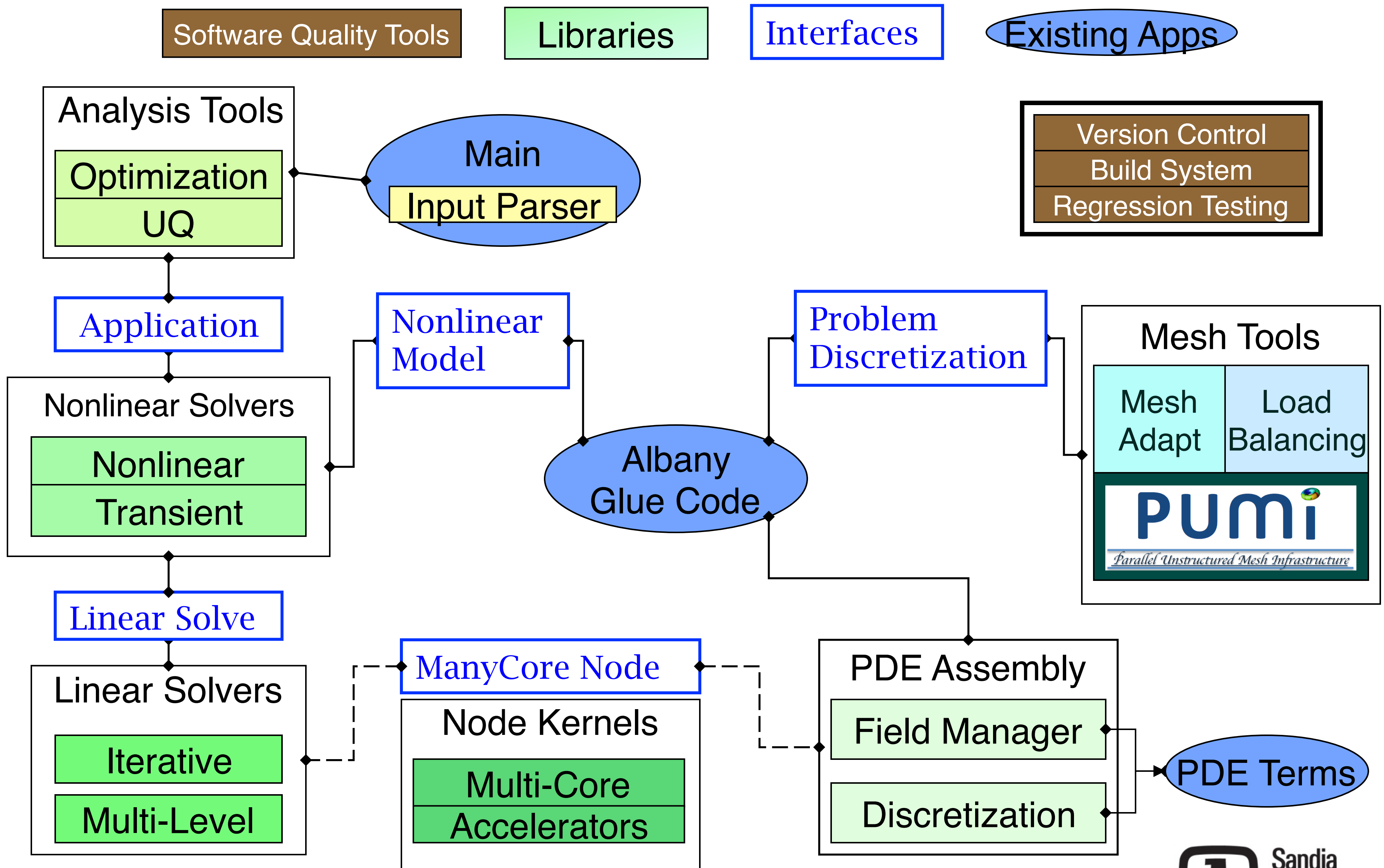


# Albany Multiphysics Code Targets Several Objectives

- A finite element based application development environment containing the "typical" building blocks needed for rapid deployment and prototyping
- A mechanism to drive and demonstrate our Agile Components rapid software development vision and the use of template-based generic programming (TBGP) for the construction of advanced analysis tools
- A Trilinos demonstration application. Albany uses ~98 Sandia packages/libraries.
- Provides an open-source computational mechanics environment and serves as a test-bed for algorithms under development by the Laboratory of Computational Mechanics (LCM) destined for Sandia's production codes

Sandia National Laboratories

# Albany – Agile Component Architecture




Sandia National Laboratories

# Agile Toolbox: *Capabilities*

Sandia National Laboratories

### Analysis Tools (*black-box*)
- Optimization
- UQ (sampling)
- Parameter Studies
- V&V, Calibration
- OUU, Reliability

### Analysis Tools (*embedded*)
- Nonlinear Solver
- Time Integration
- Continuation
- Sensitivity Analysis
- Stability Analysis
- Constrained Solves
- Optimization
- UQ Solver

### Linear Algebra
- Data Structures
- Iterative Solvers
- Direct Solvers
- Eigen Solver
- Preconditioners
- Matrix Partitioning

### Architecture-Dependent Kernels
- Mulit-Core
- Accelerators

### Composite Physics
- MultiPhysics Coupling
- Solution Control
- System Models
- System UQ

### Mesh Tools
- Mesh I/O
- Inline Meshing
- Partitioning
- Load Balancing
- Adaptivity
- Remeshing
- Grid Transfers
- Quality Improvement
- Search
- DOF map

### Local Fill

#### Discretizations
- Discretization Library
- Field Manager

#### Derivative Tools
- Sensitivities
- Derivatives
- Adjoints
- UQ / PCE Propagation

#### Physics Fill
- PDE Eqs
- Material Models
- Phys-Based Prec.

- Objective Function
- Constraints
- Error Estimates
- MMS Source Terms

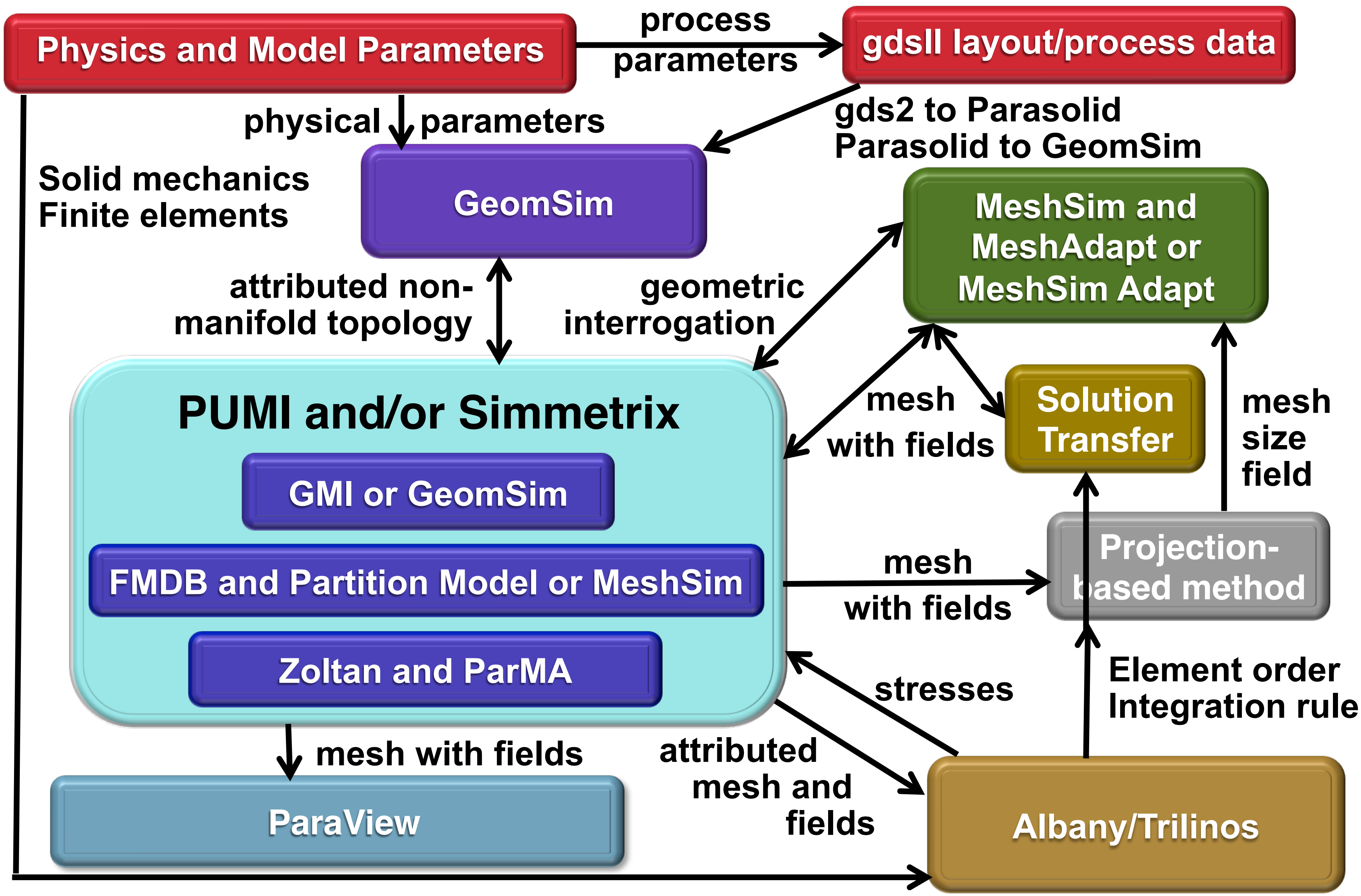
### PostProcessing
- Visualization
- Verification
- Feature Extraction
- Model Reduction

### Mesh Database
- Mesh Database
- Geometry Database
- Solution Database
- Modification Journal
- Checkpoint/Restart

### Utilities
- Input File Parser
- Parameter List
- Memory Management
- I/O Management
- Communicators
- Runtime Compiler
- MultiCore Parallelization Tools

### Software Quality
- Version Control
- Regression Testing
- Build System
- Backups
- Verification Tests
- Mailing Lists
- Unit Testing
- Bug Tracking
- Performance Testing
- Code Coverage
- Porting
- Web Pages
- Release Process

# Structural Analysis for Integrated Circuits on BG/Q



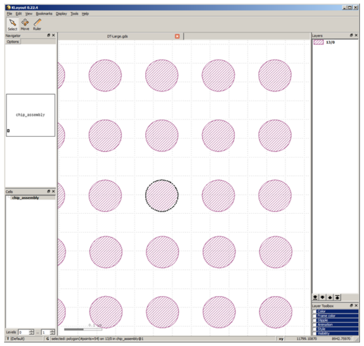
# From Design Data to Geometry for Meshing

Need complete non-manifold solid model for:

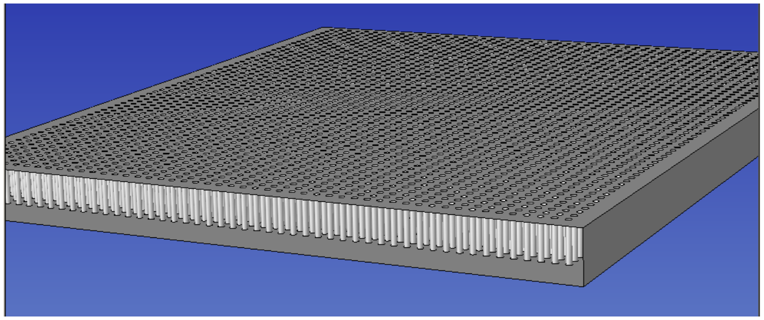
- Automatic mesh generation
- Supporting high-level problem specification
- Maintaining geometric fidelity during mesh adaptation

Tool to take design/process data and create solid model

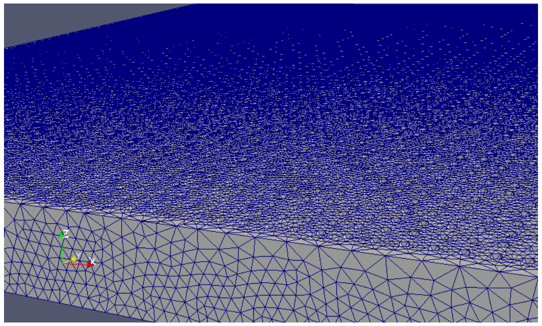
- Basic design data in 2-D layouts (gdsII/OASIS)
- 3rd dimension must be added
- Process "knowledge" critical for constructing full geometry
- Set structures and methods build solid model using modeling kernel operations



**GDS2 layout**



**Solid model – constructed from layout and process information**



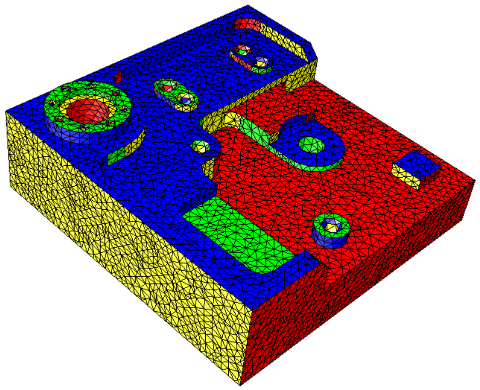
**Mesh**

# Parallel Mesh Generation

All procedures are fully automatic, user not required to partition

Surface Meshing

- Distributes model faces between processes
- Requires # model faces > # processors to scale. In practice this isn't an issue

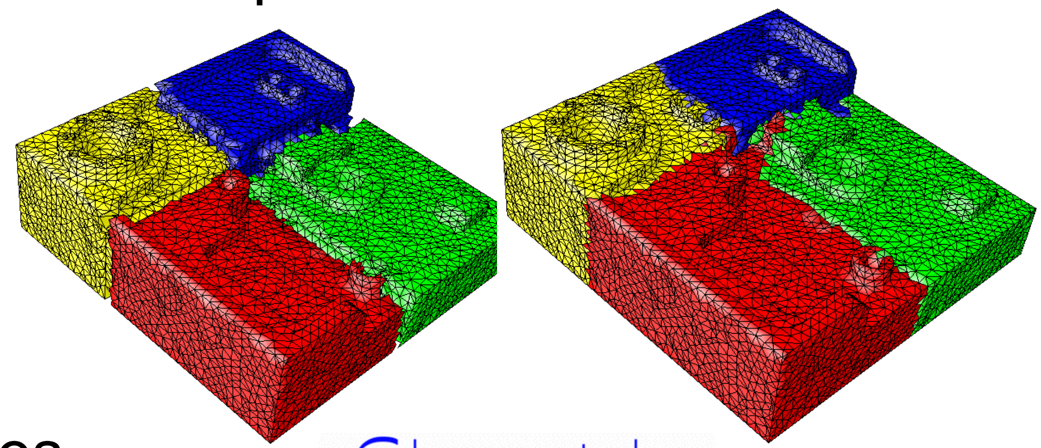


Volume Meshing

- Load balancing done through spatial decomposition
- Mesh interior to each part is created, then repartitioning done to mesh unmeshed areas between part boundaries

Mesh Improvement

- Local operations done on each part
- Local migrations done between parts to improve elements on part boundaries



Simmetrix Inc.

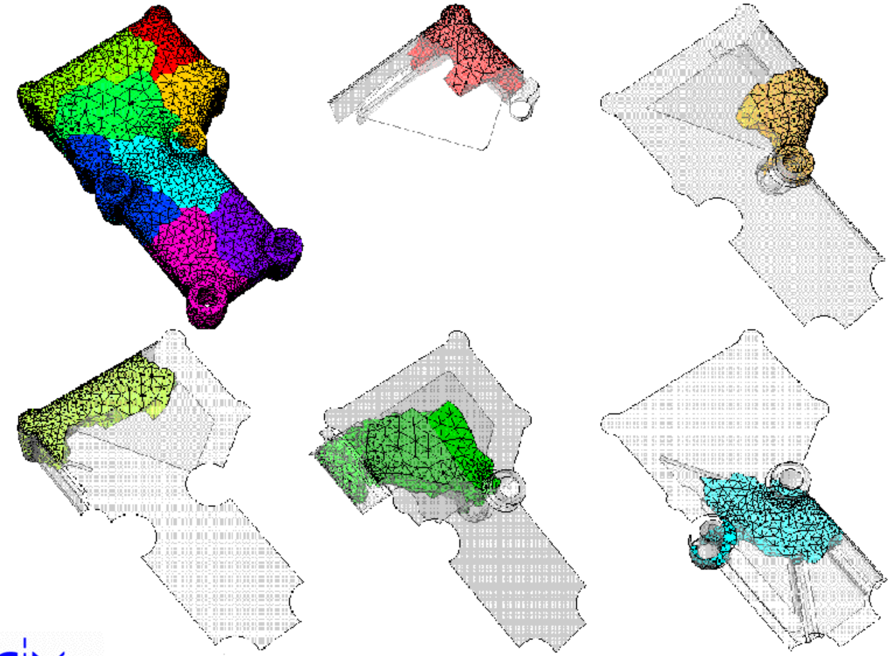
# Parallel Geometry

Problems

- CAD kernels not available on computers like BlueGene
- Even if they were, keeping full geometric model on each processor doesn't scale

Simmetrix' solutions

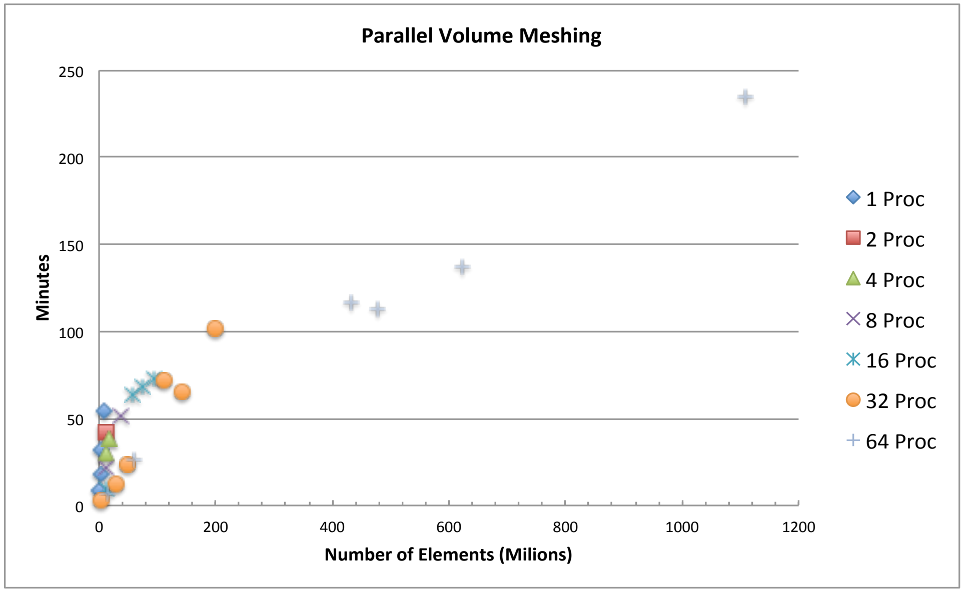
- Geometry representation that can be used anywhere
- Geometry is able to be distributed in parallel
  - Only model entities needed for mesh on each processor are on that processor. Model entities migrate with mesh
- Both discrete and CAD geometry supported

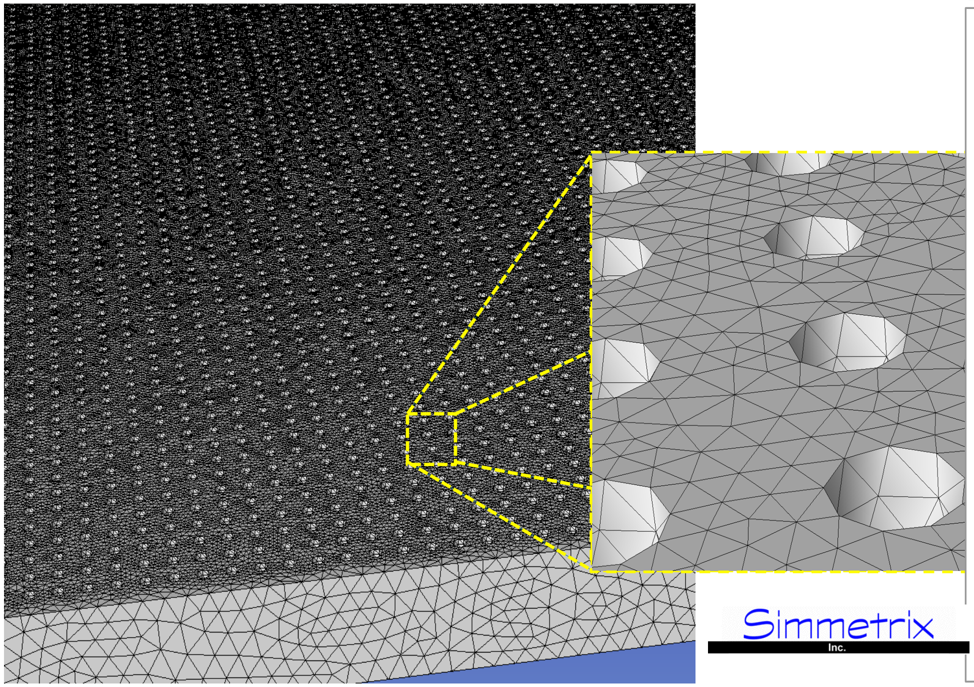


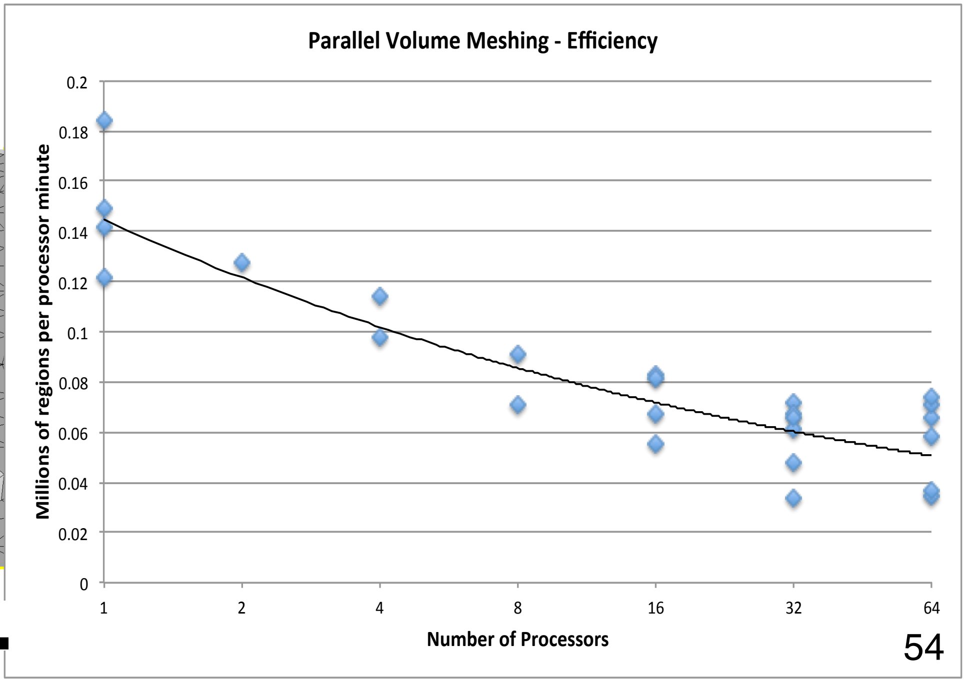
Simmetrix Inc.

# Parallel Mesh Generation Results

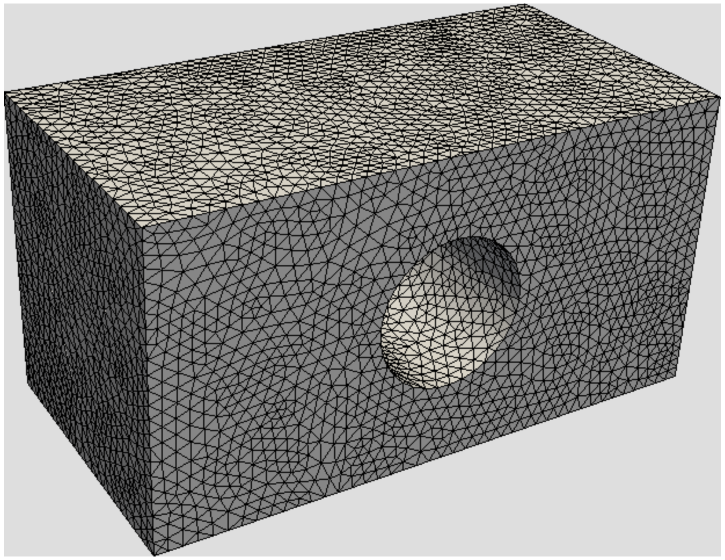
- Scaling parallel mesh generation is difficult
  - No a-priori knowledge of how to partition
  - Partitioning must be determined as meshing proceeds
- Results for volume meshing



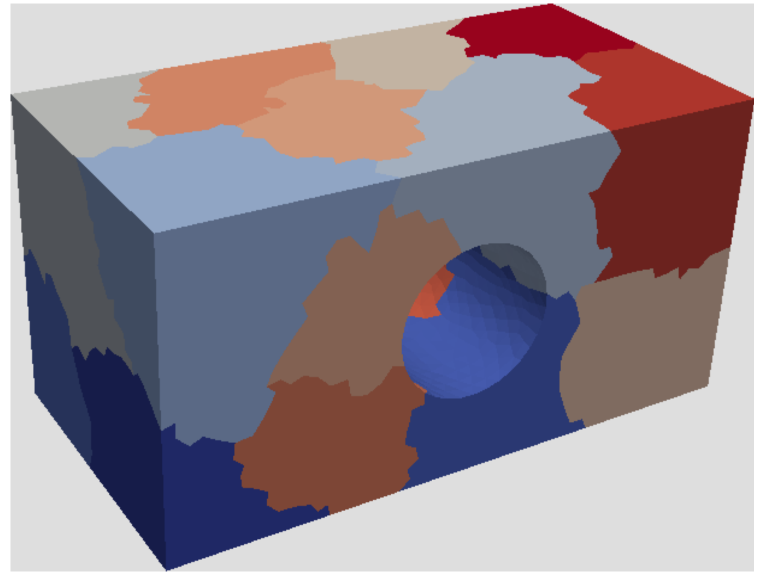



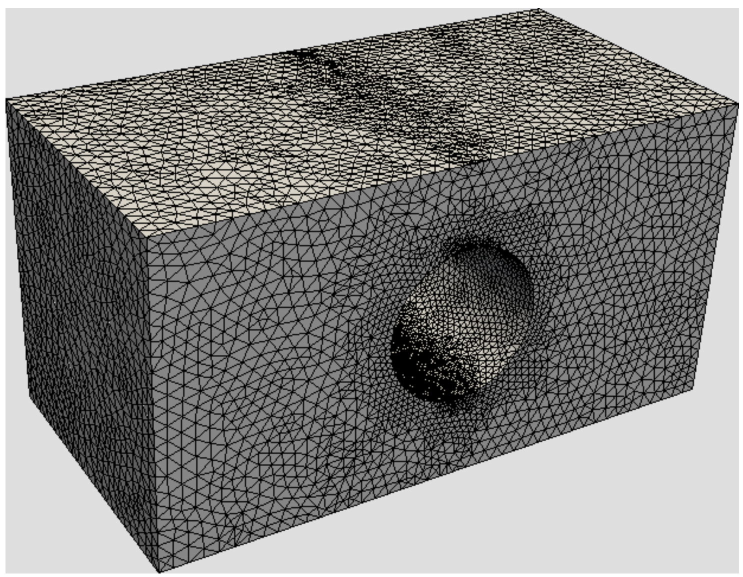
# Small Parallel Adaptive Albany Example



Initial mesh



Initial mesh partition



Adapted mesh

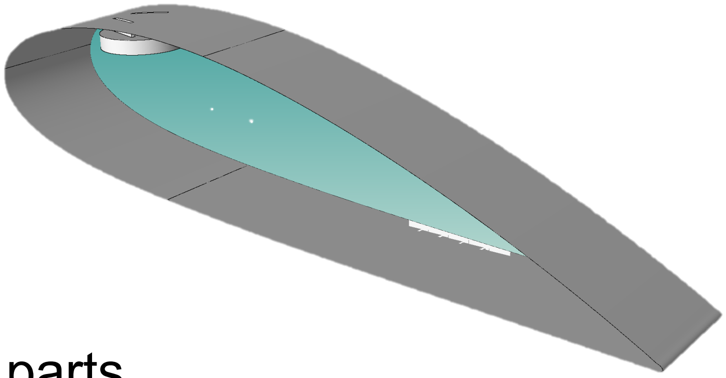
# Hands-on Exercise Outline

## Simmetrix Mesh Generation

- Video demonstrating Simmetrix mesh generation tools

## PUMI

- Air foil with actuator
  - Simmetrix GeomSim Advanced Parametric model generated from Parasolid model
  - Initial mesh has 93e3 elements and 2 parts

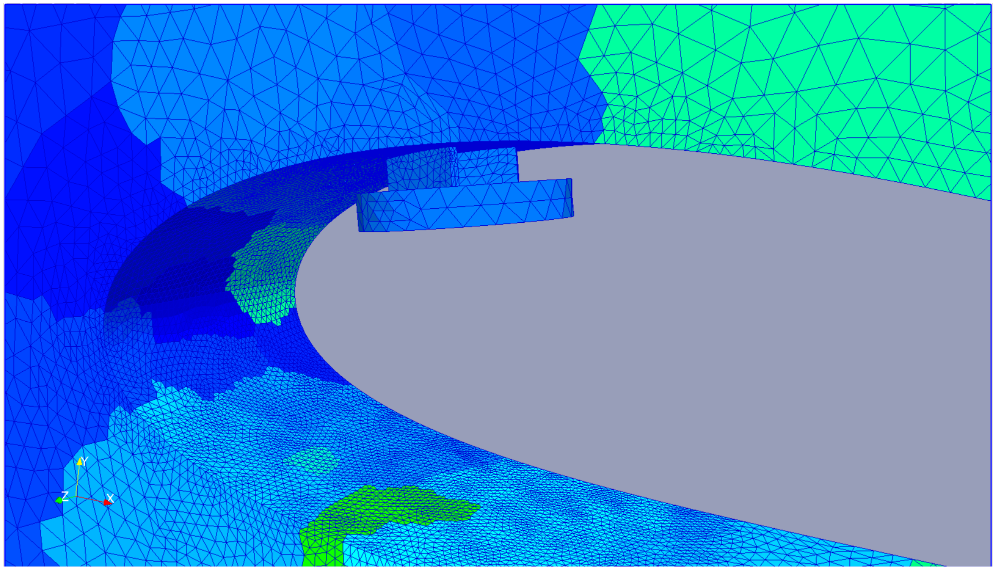


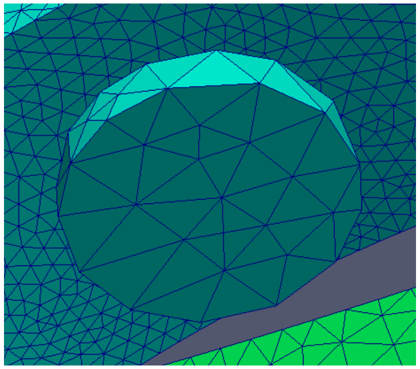
- Partition via Zoltan
  - Geometric and graph based (ParMetis)
  - Partition to 512 parts on 128 cores

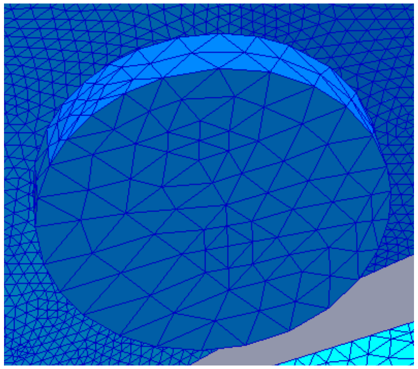
# Hands-on Exercise Outline

## PUMI (cont.)

- Parallel Mesh Adaptation
  - Adapt to 731e3 elements with tag based refinement
  - Adapt from 731e3 to 21e6 elements with an analytic size field on 512 cores
    - Predictive load balancing
  - New mesh vertices ‘snap’ to Simmetrix model
- Visualization with ParaView
- Video demonstrating mesh adaptation concepts



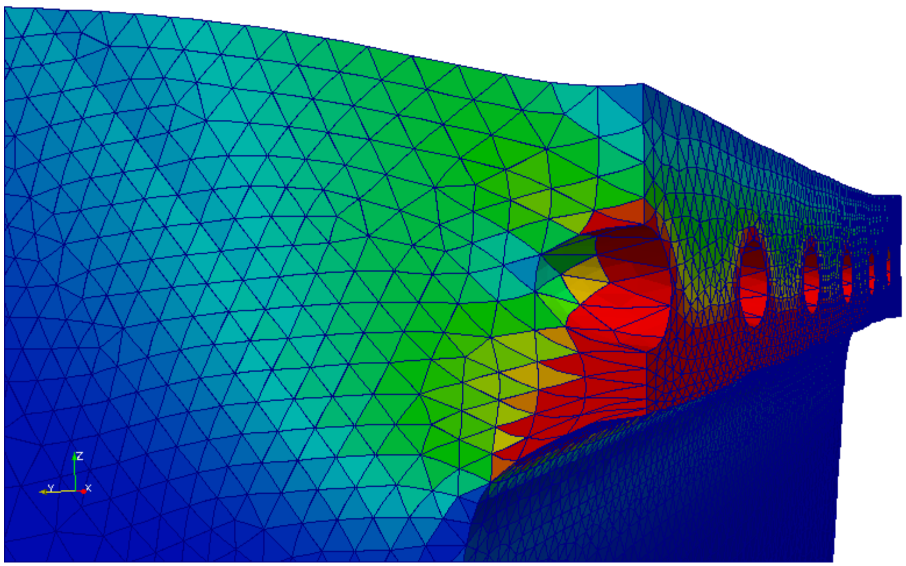


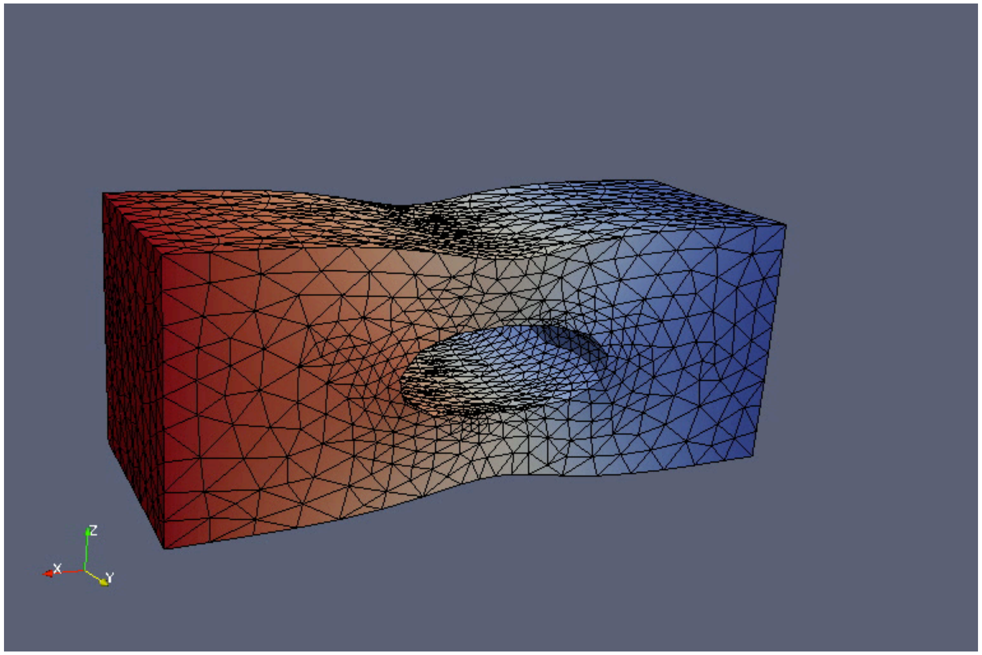


# Hands-on Exercise Outline

Albany

- Baseline parallel adaptive elasticity calculation in Albany
- Visualization with ParView
- Preconditioner control
- Adaptive elastic deformation





# PUMI: Parallel Unstructured Mesh Infrastructure

## Parallel Capabilities

- Unstructured 3D meshes w/ mixed element topology
  - Support for higher order elements
- Direct relation to geometric model
  - Parasolid, ACIS, and discrete models supported
- Solution based mesh adaptation
- Static and Dynamic partitioning
  - Integration with Zoltan and ParMA
- Ghosting
- Functional interfaces for coupling to analysis codes
  - Existing coupling with PHASTA, Albany/Trilinos, NASA FUN-3D, and SLAC ACE3P

## Download

- https://redmine.scorec.rpi.edu/projects/pumi

## More Information

- https://www.scorec.rpi.edu/pumi

# Zoltan Toolkit: Suite of Partitioners

## Capabilities

- Dynamic load balancing and static data partitioning
  - Geometric, graph-based, hypergraph-based
  - Interfaces to ParMETIS, PT-Scotch, PaToH
- Graph coloring
- Graph/matrix fill-reducing or locality-preserving ordering
- iZoltan interface supports ITAPS mesh interfaces
- Coupled to PUMI

## Download

- http://trilinos.sandia.gov

## More Information

- http://www.cs.sandia.gov/Zoltan/
- kddevin@sandia.gov

# ParMA: Partitioning Using Mesh Adjacencies

## Parallel Capabilities

- Dynamic partitioning procedures using mesh adjacencies and partition model information
  - Any mesh adjacency can be obtained in O(1) time (assuming use of a complete mesh adjacency structure).
- Partition improvement to account for multiple entity types
  - Improved scalability of solvers by reducing peak entity imbalance(s)
  - Avoid graph construction – low memory cost
- Predictive load balancing for mesh adaptation
  - Avoid memory exhaustion
- Coupled with PUMI

## Download (as part of PUMI)

- https://redmine.scorec.rpi.edu/projects/pumi

## More Information

- https://redmine.scorec.rpi.edu/projects/parma

# MeshAdapt: Unstructured Mesh Adaptation

## Capabilities

- Parallel adaptation of unstructured 3D meshes w/ mixed element topology
- Supports general changes in mesh size including anisotropy
  - Typically driven by a solution field based size field.
- Can deal with any level of geometric domain complexity
- Can obtain level of accuracy desired
- Solution transfer can be applied incrementally
  - Callbacks for application defined transfer procedures.
- Coupled with PUMI

## Download

- https://redmine.scorec.rpi.edu/projects/pumi

## More Information

- https://www.scorec.rpi.edu/meshadapt/

# Albany: Multiphysics Simulation Environment

## Capabilities

- A finite element based application development environment for rapid deployment of analysis capabilities.
  - AgileComponents and TBGP enables rapid application and feature development
  - Linked to Trilinos linear and nonlinear solvers for scalability
  - AD Jacobian, derivatives for SA and UQ
  - LOCA for continuation, stability analysis, bifurcation tracking
  - 160+ example physics applications in test suite

## Download

- https://software.sandia.gov/albany

## More Information

- Glen Hansen [gahanse@sandia.gov]
- https://software.sandia.gov/albany/gettingStarted.pdf
- https://redmine.scorec.rpi.edu/projects/fmdb/wiki/Building_Albany_and_PUMI_from_Scratch

# Closing Remarks

A set of tools to support parallel unstructured mesh adaptation have been developed

- Parallel mesh infrastructure
- Dynamic load balancing
- Mesh adaptation
- Support for heterogeneous parallel computers under development

Tools used to develop parallel adaptive simulations

- Both unstructured mesh finite element and finite volume procedures being developed
- Multiple problems areas – CFD, MHD, EM, solids
- Can account for semi-structured mesh regions, evolving geometry, high order curved meshes

More Information: shephard@rpi.edu